UNIVERSITÀ
DEGLI STUDI
DI PADOVA

Master's Degree in Computer Science

Academic year 2023/2024

MOBILE PROGRAMMING AND MULTIMEDIA

Prof. Ombretta Gaggi

Written by Michael Amista'

Contents

A. Mobile Programming **6**
1 Introduction **6**
2 Cross-platform framework **10**
2.1 Introduction to cross-platform development 10
2.2 Raj & Tolety classification 11
2.3 El-Kassan classification 16
2.4 Energy consumption analysis 18
2.5 Flutter 20
2.5.1 DART 21
2.5.2 Architecture 22
2.6 React Native 25
3. Store deployment **27**
3.1 Google Play Store 27
3.2 App Store 30
3.3 After the deployment 33
4. Mobile Design **36**
4.1 Design a good interface 36
4.2 Gestures 42
4.3 Other type of interactions 47
4.4 Emotional design 47
4.5 Putting all together: the app as an overall narrative 49
B. Multimedia Data **51**
1 Introduction **51**
2 Media classification **53**
3 Information Compression **55**
3.1 RLE 56
3.2 Shannon-Fano algorithm and Huffman Code 57
3.3 LZW 59
4 Images **63**
4.1 Bitmap (BMP) 65
4.2 Graphics Interchange Format (GIF) 66

4.3 Portable Network Graphics (PNG) 67
4.4 Joint Photographic Experts Group (JPEG) 68
4.5 JPEG 2000 73
4.6 Vector Graphics 75
5 Audio 76
5.1 MPEG-1 81
5.2 MIDI 85
6 Video 88
6.1 Motion JPEG 90
6.2 MPEG 93
6.3 MPEG-2 95
6.4 MPEG-4 95
6.5 MPEG family 97

Course introduction

The course is divided into two different sections:

A. **Mobile Programming** (slides of this part will be updated strictly before every lesson)
B. **Multimedia Data** (representing, encoding, compressing and digitalize different types of data, such as text, audio, color, etc.)

The course will cover the main technologies for encoding, storage and transmission of multimedia information.

Multimedia is a fundamental aspect to consider when talking about mobile programming since most of the apps use voice messages, images interchange and other multimedia communication → the available memory of devices, after memorizing all this data, decreases over time. It is important to study how to manage this data, for instance: "I want a fast encoding which reduce the storage space needed and also consider the battery constraint".

The first thing of a well-designed application is the application does what it should do (e.g. train application: "I have to be able to buy tickets in a little amount of time if the train is leaving".

There are a lot of applications in the market, it is difficult to find a new useful idea, but it is possible to learn **how to develop applications which are fast, easy to use and that do what they are designed for**.

Find a good metaphor is a key factor in developing mobile applications.

When we develop mobile applications, we must consider all the other mobile scenarios services (e.g. phone calls) that can interfere with the application. So, take care about the common usage of devices on which applications run.

Are there differences between these images?

Not so much, the quality can be maintained even with less storage space!

It is possible to remove all the image information which we cannot see to save space. We will see how this work.

It is possible to manage **images** using some properties, such as size, quality, transmission, visualization. There are a lot of file formats we can use: GIF, PNG, JPEG, JPEG2000, etc.

It is possible to manage **sounds** using some properties, such as fidelity, transmission, playback, etc. There are a lot of file formats we can use: WAV, MP3, etc. Even in sounds there some parts of the file we are not able to listen, so it is possible to remove them to save space.

It is possible to manage **videos** using some properties, such as quality, representation, transmission. There are a lot of file formats we can use: H261, H263, MPEG family, DivX, Xvid.

There are some reasons behind data compression (storage space, transmission time). Different types of data compression, for instance *lossless* vs *lossy* compression/encoding.

We will see different **cross platform frameworks** (pros and cons) and the main elements that define the quality of mobile applications. The market is highly segmented, so it is crucial to develop application in a sort of "virtual machine" which enables to abstract the application development from the single device.

Target skills and knowledge: mobile interface design, cross platform development, emotional design, wearable devices, market.

HOMEWORK

4-5 homework during the course: it is possible to avoid the two oral questions at the exam submitting all the homework and discussing them during classes. The typical outcome of homework is "passed" or "not passed".

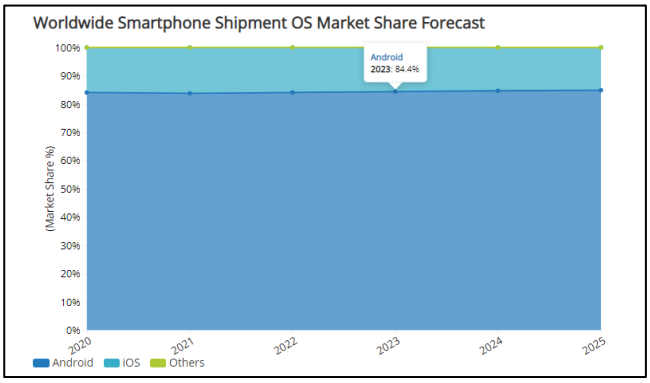
A. Mobile Programming

1 Introduction

Web application vs mobile application, which choose? We'll answer this question studying users' targets. We must make choices according to users' needs.

In our days, most people use smartphones, around 5.44 billion. The smartphones market is enormous, it is not just the smartphones' sales but even the application market, all the tools developed and available on those devices.

Statistics: China, India, and the United States are the countries with the highest number of smartphone users. US used most desktop device.



Android and iOS are the main operating systems available for smartphones. Android is used by 84.4% of smartphones. "*If I have to develop an application for just one platform which one, should I choose?*" iOS, since there the market is richer than Android which is also **more fragmented** → not guaranteed the application work in the same way for all the Android smartphones.

Most of users use mobile devices (most use smartphones) to visit websites. When we need to decide to develop an application for desktop or mobile, in reality we should take aspects from both.

NOTE: when we will talk about mobile devices (phones, tablets, wearables, etc.), we will refer just to mobile phones.

Tablets' market is not so rich and consider developing applications for tablets is quite different from smartphones (e.g. two fingers usage, the mechanisms to use tablets are different and due to this it is not so easy to develop an application for both smartphones and tablets).

Just two regions where the desktop is used more than smartphones: North and South America (about 60% vs 40%). In Europe, Asia and Africa smartphones are more used.

When we talk about users, we typically refer to the whole population but there is a distinction between female and male: for instance, female use more mobile phones. We need to consider this when we develop an application, understanding even its genre direction in the market.

There are differences depending on which time of the day we use technology.

Statistics: In 2018 Italy had the third place in the world, with 34 million people. Time spent on social networks every day, considering all the devices, is 1 hour and 53 minutes, and 2 hours and 20 minutes every day spent on the Internet using a smartphone.

The application must deal with all the users' activities on mobile devices for instance: there are push notifications, advertisements, and other type of interaction.

Mobile phones are not considered anymore as a simple device to make calls, but incorporate a lot of different features:
- Messages, calls
- Internet navigation
- Sensor data collection and usage (app for training, biking, running, etc.)
- Agenda
- Entertainment (games, music, video, reading, etc.)

All these features are provided by apps and not by the mobile device itself!

False myths:

1. Mobile app development is not expensive: different platforms on which develop, different users' habits to consider.
2. Mobile app is easy.

⇨ Mobile app development requires **big teams**: who write the code, who study the users' habits and so on.

Application or responsive website?

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**
  This depends on the scenario on which we work: for instance, if we need to park a car (with an app to pay) and then go to an appointment, an app is not so appropriate since the app require an installation, login phase and so on → waste of precious time because we may be late.

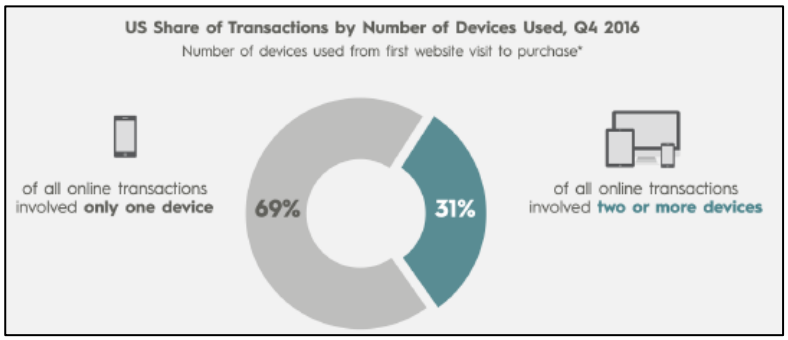
  NOTE: in this scenario, the app is useful if we often use the same parking → user' information already memorized in the app → faster process to buy the parking ticket.

Typically, when a user uses a lot a service the app is preferable, on the other hand if the service is seldom used the website is preferable. The set of both modalities is always preferable.

- **Native interface vs. Company brand**
- **Development time**
- **User interaction** (ex: push notification, gestures)
- **Access** (a good icon is the key)
- **Target** (loyalty vs. reach)

Def. "Conversion rate" typically refers to the percentage of users who take a desired action, such as making a purchase or signing up for a service, out of the total number of users who visit a website or use an app.

Desktop has a conversion rate which is the double than mobile phones: a higher proportion of desktop users are completing the desired actions compared to mobile users.



There are scenarios where **users start an action on the mobile and conclude it on the desktop** → there must be an effective communication between the two versions of the service (e.g. if I put an item in the mobile basket, I must find it in the desktop basket).

e-commerce apps: typically, iOS customers spend more on products, even if the percentage of iOS devices' distribution is less than the Android one. This is an example of studying the users before developing an application.

The App must be designed and developed in a high quality. There are millions of apps in the stores: 25% of these apps are used only once. Typically, once an application has been removed the likelihood of reinstalling that app decreases.

A user usually spends 89.2% 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.

All this data helps to understand which are the interesting aspects of developing an application which is an expensive work, it has to pay!

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** (note: previous car parking scenario).

When to create an app

- A lot of graphics or calculations
- Camera, sensors, or microphone usage
- Gallery or contacts' access (higher acceptance rate on mobile)
- Push notifications or background service
- For games
- It is the only way to have access to the store

Resources

The development of a mobile app requires several resources:

- Interface design
- Development (e.g. Which operating system?)
- Maintenance

Web advantages

- They require a very low knowledge base, HTML is popular and 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 (Google Play, App Store):
    - Apple can require more than 2 weeks!

App vs Desktop applications

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

2 Cross-platform framework

2.1 Introduction to cross-platform development

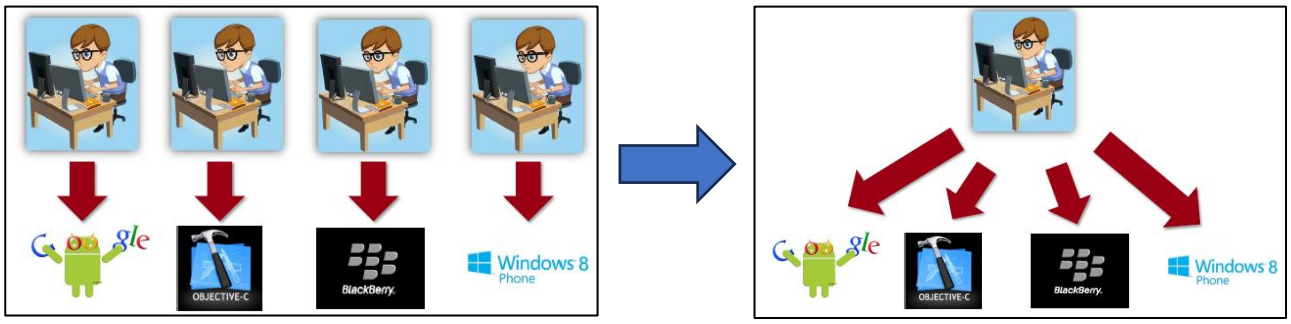
PROBLEM: which platform and framework? The framework and platform should be chosen according to what we need to develop, we will see how to choose them properly.

Statistics: over 3 smartphones, 2 are Android devices and 1 is iOS.

Be careful: Android devices are more than iOS, but the Android market is more fragmented → possible problems: different UI behaviour, different device features (develop flexible layout).

We have **different OS with different languages**: it is necessary to develop different apps for several devices. This requires a lot of knowledge about: operating systems, programming languages, development tools (IDE, simulators, etc.), API, sensors/equipment (may differ in value range or other features), screen size, computational capacity, etc.

⇨ We need to be careful about all these features! Cross-platform frameworks try to remove some of these constraints. The idea is to **develop once and then change it a little bit according to the device** (e.g. some APIs may be available just on iOS, others only on Android → deal with that).



Main features of cross-platform frameworks:

- Application developed on time, using only one programming language (or a set of languages).
- 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.

Not always the most known frameworks represent the best solution. We will analyse some data to discuss about which framework should be chosen according to the involved scenario.

Today the most important applications have been developed with cross-platform frameworks, the native approaches are not so used. The most advantages in using frameworks can be found in reduced costs and time saving.

Cross-platform mobile development	
pros	**cons**
wide market reach	possibly slower performance
single codebase	UX and UI discrepancies
faster and cheaper deployment	
reduced workload	
platform consistency	

Key points:

- Frameworks affect a lot the performances of your application.
- Not always frameworks allow to use native UI but provide their icons, widgets, etc.

Pros of native applications:

- 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 → Flutter now supports a lot of APIs).
- An Apple computer is always needed.

Cons of native applications:

- Fragmentation = higher development costs.
- Problems with test.

Steps for app development:

1. Idea analysis (who is the target?, what should the app offer?)
2. Interface design
3. App development
4. Store deployment

⇨ Store deployment is necessary every time there is an update and for each platform.

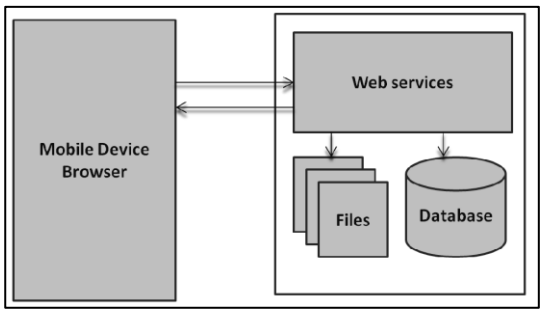
⇨ **Native development requires repeating steps 2-3-4 for each platform**.

2.2 Raj & Tolety classification

Frameworks' classification is still an open problem. Raj and Tolety classification, the most used in research areas, define 4 different classes:

1. **Web** Approach
2. **Hybrid** Approach
3. **Interpreted** Approach
4. **Cross-compiled** Approach

WEB APPROACH



Frameworks that allow to develop web applications and <u>not</u> mobile applications. Those frameworks allow to use devices' APIs to have access to the device's features.

(Mobile application always means we have an icon by which we can interact with the app, not in this scenario).

⇨ **Pros**:

- Since the mobile web browsers are fairly standardized the Web User Interface can be reused across different platforms.
- No installation necessary (just write the URL on browsers).
- Easy update (every time the user writes the URL on a browser, or whenever he reloads the page, he gets the latest version of the application → no store updates to manage).

⇨ **Cons**:

- No store publishing.
- Network connection necessary. The network constraints may lead to inferior performances due to connectivity issues or network delay.
- Difficult test (these frameworks does not provide a simulator; the application can run on so many different combinations and devices → we cannot test all of them).
- Strongly connected to HTML5 support of the device.
- **Non-native interfaces bring to low usability**.
- A web application is limited to leverage the gestures offered by the platforms.

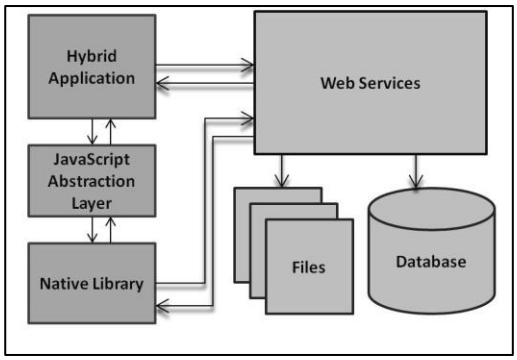
<u>A web-based approach</u>: **Progressive Web App** (PWA) refers to web pages that <u>behave</u> like native applications. Progressive Web App represents the first framework to leverage the knowledge of web developers (HTML, CSS, JavaScript). In particular:

- They are developed using web technologies, therefore HTML5, CSS3, JavaScript.
- It works independently from the browser, using ***progressive enhancement*** (the more features the browser provides, the more features the app provides).
- **It works even offline** (cache), but with limited support.
- **Can be installed without using the store** (but in this case, they are a sort of link) + icon.
- 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** (with their limits, not effective as the app notifications).
- No need for stores to publish the app, but there is no payments management, and there is no control of what is published.

Frameworks that use this approach: Sencha Touch, React (no React Native), Angular, jQuery.

If the webapp looks like a native application, it means that it uses the hybrid approach.

HYBRID APPROACH



The application uses the engine of the browser and the native library to seem more like a native app. The browser engine renders and displays the HTML content in full screen web view control.

The device capabilities are exposed to the hybrid application through an abstraction layer.
The abstraction layer exposes the device capabilities as JavaScript APIs.

This approach encapsulates standard web technologies in a native wrapper that enables to distribute the application as a native app through the different stores.

Main difference from web approach: here we are using only the engine of the browser and not the browser itself (= here we do not see the address bar and the browser interface but just the content of the page).

⇨ **Pros**:

- Store publishing available.
- Reusable UI across different platforms utilizing native platform features.
- Usage of device components.

⇨ **Cons**:

- **Lower performances** ➔ 2 components running: code + web engine.
- Even if it is so simple to recognize that the app hasn't been developed with its native language, UI does not follow the native Look and Feel so specific styling might be required.

Example: PhoneGap/Cordova

It was a project started in 2008 to solve problems as:

- Development of mobile applications using web technologies.
- Solve the problem of low support of mobile browsers to HTML5.
- Allow access to different features of the device.

Actual support to HTML5 of the mobile browsers and the HTML5 evolution have partially solved these problems.

Apache Cordova framework (the new name) is a hybrid framework where applications development works with HTML, CSS and JSS (these languages are already well known by all web developers). 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.

```
<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>
```

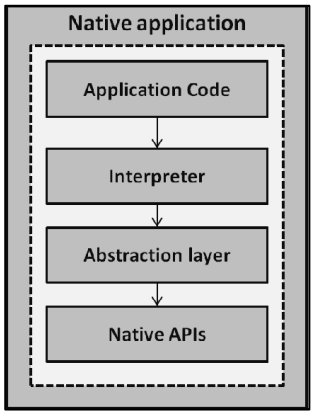
A simple instance of the framework.

It is possible to access to device's features by adding an event handler that waits until the library, used to access to device's APIs, is loaded. Then the developer has access to camera, sensors, etc.

Cordova usually is not used stand-alone, but as a support framework for other frameworks (e.g. Monaca, Framework7, NativeScript, Ionic Capacitor, Progressive Web Apps).

Be careful about performances: the framework decreases the learning curve for web developers, in fact web technologies used also for mobile, but I pay this in performances → simple applications can easily run (e.g. train timetable) but more the application is complex more difficult becomes to run. So, the framework can be interesting if the application is not so heavy.

INTERPRETED APPROACH



The **application code** is deployed to the mobile device and gets interpreted thereafter.

There is an **interpreter** which executes the code at runtime across different platforms and thus supports cross platform application development.

The interpreted application interacts with the abstraction layer to access the **native APIs** (native features).

Apps developed using HTML, CSS, JavaScript but in a different way from Cordova: this framework creates all with JavaScript, all the entire pages are generated by JavaScript (tag, etc.) and then translated into the using platform by an interpreter at execution time.

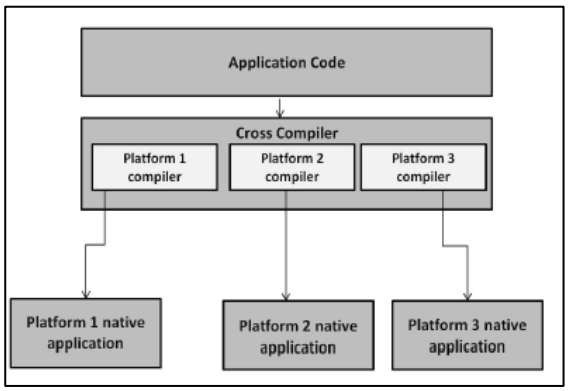
⇨ **Pros**:

- Native Look and Feel.
- Store publishing.
- APIs for device components.

⇨ **Cons** (depending on the chosen framework):

- Really difficult to reuse the UI, it depends upon the framework abstraction level.
- Available features depend on the framework.
- The interpreter can have low performances → 2 components running: app + interpreter.

CROSS-COMPILED APPROACH



The cross compiler is responsible for generating the executable code for a particular platform.

The developers can write the source code in a common programming language and the cross compiler compiles the source code into native code.

➔ This whole approach is dependent upon the efficiency and reliability cross compiler.

In this case it is really hard to distinguish native app from app written using a cross-compiled framework (e.g. React Native, Flutter → the output is really good.).

⇨ **Pros**:

- Cross compiled application provides all the features that the native application provides.
- Native interface.
- Good performances.
- Store publishing.

⇨ **Cons**:

- UI cannot be reused. Also, the platform specific features such as camera access, location services, local notifications, etc., cannot be reused. These features are platform specific and the way to access these features varies from platform to platform.
- Very complex apps can have problems during the building process.

Frameworks that use this approach: React Native, Flutter, Solar2D.

2.3 El-Kassan classification

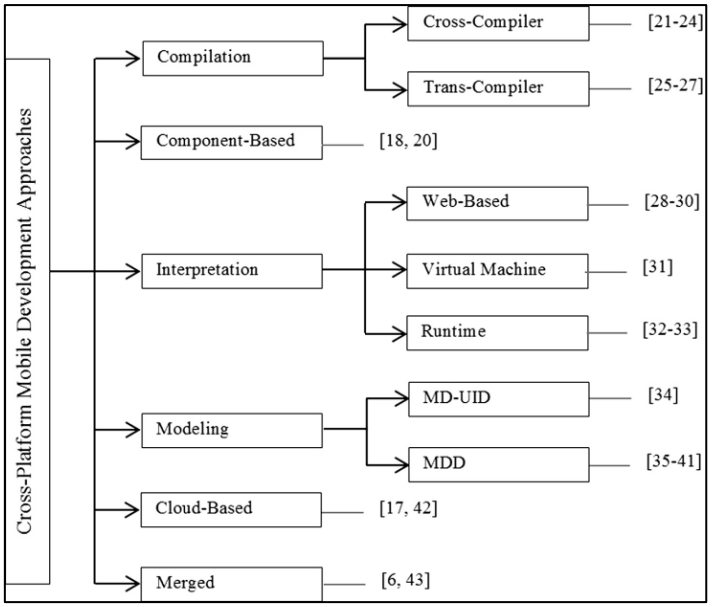
There are several methods for frameworks classification, some of them based on the **development approach**, others **based on the result** (the final output).

El-Kassan et al. divide the app's output into three categories:

- **Native app**, not necessarily an app developed using a native language (swift [iOS], java [Android]) but that looks like a native application.
- **Web app**, the application run using the browser engine. Not really an application because we need to enter a URL in a browser.
- **Hybrid app**, a middle approach: app that requires other components to run. Hybrid (browser engine + code) and Interpreted (interpreter + code) approaches produce a hybrid app.

Choosing one approach than others depend on the scenario (e.g. if the app has to be synchronized, for instance, every second I should choose a native app).

Another classification of El-Kassan about *Cross-Platform Mobile Development Approaches*:



<u>COMPILATION approach</u>

The common use of the compiler is to transform the source code written in high-level programming language to an executable program. For cross-platform development there are two types of approaches: ***cross-compiler*** and ***trans-compiler***. The difference between those two sub-approaches is in the management of the source code during the compilation process.

- **Cross-Compiler**

  This approach translates the source code directly into executable code (native code) for the specific platform. Cross-compiler approach <u>does not require native development tools</u> during the compilation process.

- **Trans-Compiler**

  This approach, differently from the previous one, firstly translate the source code (e.g. JavaScript) into another high-level programming language for the specific platform (e.g. Java for Android o Objective-C/Swift for iOS), then it compiles the translated code. Trans-compiler approach requires native development tools during the compilation process, native development tools such as: Android Studio, Xcode, generation of executable binaries files, usage of native libraries and all other types of native tools useful to correctly complete the compilation to get the final app before its deployment.

NOTE: Flutter uses both cross-compiler and trans-compiler approaches, in fact it is a developer choice to choose one or the other. If the developer chooses the trans-compiler approach, the source code is translated into Java for Android (an "Android" folder is created) or into Objective-C/Swift for iOS (an "iOS" folder is created). Then those two folders contain the code runnable in Android Studio and Xcode to build the application in both OS.

Why we have both options? Sometimes developers may need to run (or test) the app on the final platform, rather than on the simulator provided by the framework itself, because some native features may be not supported by the framework simulator. To run an app on the specific platform native development tools, such as platform's IDE, are necessary.
Another reason is that sometimes using a cross-platform framework may have an impact on app performances so, it is possible to develop just the UI with the framework (1 time) and the logic in a native way (2 times) using so the trans-compiler approach → CODE FORKING scenario.

COMPONENT-BASED approach

App development starts from different components that communicate together using interfaces (e.g. React Native). Each component has the same interface in every platform (the output is a native application), but different implementations.

Advantages: It simplifies the development, assuming that there are several off the shelf components available.

MODELING approach

Similar to the component-based approach but with this approach the developer uses several abstract models to describe user interface and functions of the application. Models are then translated into native source code.

CLOUD-BASED approach

With this approach, all the app computations 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
- No need for specific hardware

2.4 Energy consumption analysis (By O. Gaggi, M. Ciman, N. Gonzo)

Energy consumption is a crucial element for application success: apps that drain the battery are rapidly uninstalled by users.

We considered the energy consumption of apps that acquire data from different sensors:

- Accelerometer
- Compass
- Microphone
- GPS
- Camera

And **compared these results between native apps and the ones developed using cross-platform frameworks**.

Several authors measured energy consumption of mobile applications and compared the results differentiating between native and cross-platform ones.

- Thompson and other authors proposed a model driven approach for energy consumption, which require estimating the requirements before the application development.
- Other methods were used, such as applications (AppScope, Yoon…) which are used to estimate the energy consumption of each hardware components.

Measurement system: Monsoon Power Monitor → provided data: energy consumption, average current and consumption, estimated battery duration, etc.

Goal: energy consumption comparison of different hardware components during data collection, considering different platforms and different frameworks (in this experiment: one framework for each cross-platform development → Raj & Tolety).

Applications considered (sensors depend on the API available with each framework):

- 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

Note: if a framework provides different development languages, it is interesting to investigate which language has the best performances (typically we can find this information inside discussion forums).

The results showed that a **cross-platform framework is usually linked to a higher energy consumption, even if the framework generates native code**. The most expensive task is the interface update, but also data acquisition strongly influences the energy consumption. But

this is not a cross-platform only problem, since **frameworks showed different consumption depending on the operating system where they were running**.

As a summary of what said so far, we can state that cross platform frameworks consume more energy, hence creating lower performances and lower user acceptance. For this and other reasons, hence, native development should be preferred at times, since the framework choice may be too critical, or because for complex applications efficient frameworks are still missing. Titanium seems to be the framework with better consumption for the moment, but we should always consider that providing a bad application is worse than not providing an application at all.

Prof tip: search for applications developed with a certain cross-platform framework and test those applications to see how they work / how much resources they consume / etc.

Cross-compiler approach seems to be the most promising one: the key is developing an effective compiler, able to translate the source code into native one in an effective way.

Before Flutter, Prof opened a small parenthesis on Solar 2D framework, giving some examples and explaining that Solar 2D represents a good solution if we need to develop a mobile gam; in fact, the development is simpler compared to other frameworks especially for the animations management (see old Solar2D slides for more details).

2.5 Flutter

Flutter is an SDK for mobile devices, developed by Google, for the development of native application for iOS and Android starting from a unique codebase.

⇨ **CROSS-COMPILED** approach.

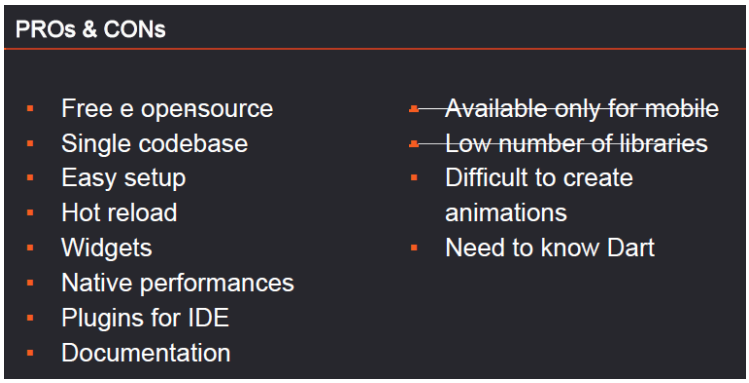
⇨ Application written in **Dart** (similar to C#).

⇨ Supported platform: Android, iOS, Android even for IOT devices.

⇨ Showcase: Google Ads, Greentea, AbbeyRoad Studios, Alibaba.com, Reflectly, etc.

Main characteristics:

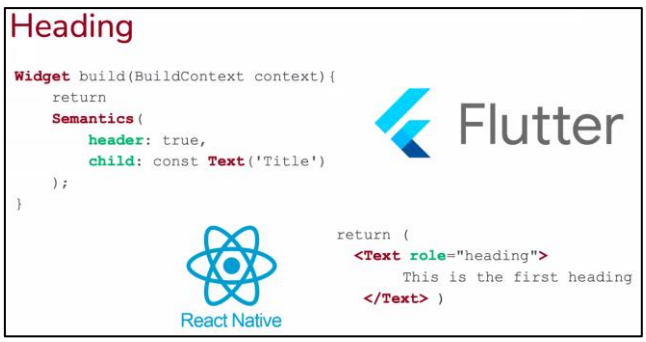
- **Fast development**: it allows to build and reload the code during runtime reducing so the development time → furthermore, Flutter build and reload just the modified code and not the entire application.
- **Expressive and flexible UI**: in the Android platform this is crucial since its market is more fragmented than the iOS one → the interface must be not just native in look but also in the feal.
- **Native performances**: widgets incorporate all the main characteristics.



No more available only for mobile: Flutter enables to develop webapp but it is not so suggested, while for Android and iOS works well due to the lot investments done in reaching native performances.

Flutter support accessibility: big fonts, screen reader, contrast → it works differently from the accessibility that we know from the course "Web technologies". For instance, the <header> tag is created as a widget that gives to this component the semantics of an <header>.

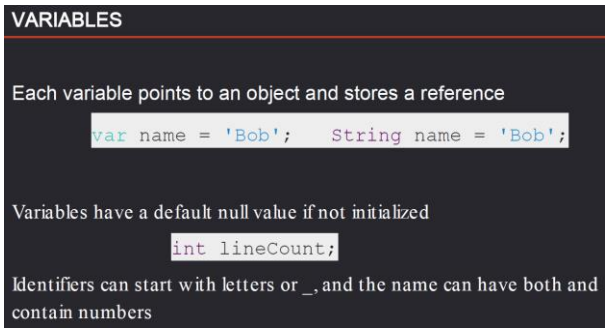
The result is that Flutter support accessibility, but we need to add more code → React Native requires less code than Flutter (look for the next example to create the <header> tag).

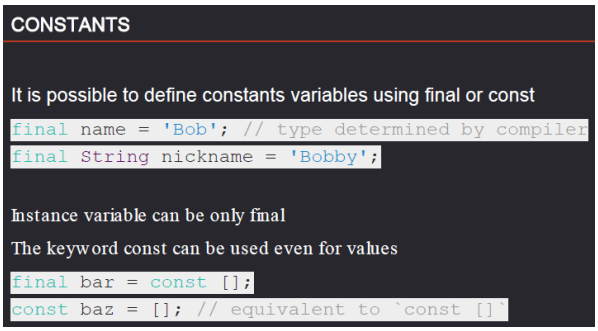


2.5.1 DART

It is a programming language, object-oriented, used to develop web, server, desktop, and mobile applications, developed by Google (first name was Dash).

SUPPORTED TYPES: Numbers (int or double, num subtypes), Strings (String), Booleans (bool), enum, List, Sets, Maps, Runes (to use Unicode characters in a string), Symbols, Generics (ex: List<type> o List<dynamic>).





LIBRARIES: every DART app is a library, and we can use other libraries to build an application.

Lazy loading for libraries: when you import a lot of libraries in your application, they are not loaded until we need them → due to this, there are situations, such as first app load, where the app can require more time to load because all these libraries are loaded together.

It is possible to use the keywords *show* and *hide* to manage the library visibility (e.g. it is possible to use just some parts of the library).

Exceptions are not managed so we need to use the classic *try catch* syntax.

Dart code can be compiled in different ways:

- **just-in-time (JIT)**: used for during development and testing time. JIT provides just an input for the simulator where we can test the app and not the final APK.
  Beyond the simulator tool, it is even possible to test the app directly on a mobile device which must be always connected to a laptop during the testing. This approach requires the app installation and an interpreter that allow to run the app.
- **ahead-of-time (AOT)**: this approach makes framework cross-compiled. AOT approach is used during deployment time, when we need to publish the app on the stores. AOT allows to get the native code from Dart, removing all the stuff that were used during the development time and which are no more useful.

2.5.2 Architecture

Two main components:

1. **Framework (Dart)**: it provides all the components to develop an application.
2. **Engine (C++)**: it is the run time environment which runs the application. Apparently, using an engine, according to Raj & Tolety classification, seems to be more an interpreted approach but in fact the engine is not used by the mobile device itself → it is used just during the development phase.

   Instead having a different compiler for each platform, Flutter provides an engine, which works on more platforms, able to translate Dart code into executable code but again just during the development.

   NOTE: when we get the final app, we use a cross-compiled approach (AOT compilation).

Framework components:

- **Material & Cupertino**: implements widget, Material (Android) and Cupertino (iOS) style, which have the native look and feel.
- **Widgets**: implements generic widgets (e.g. a text area).
- **Rendering** : simplify layout management.

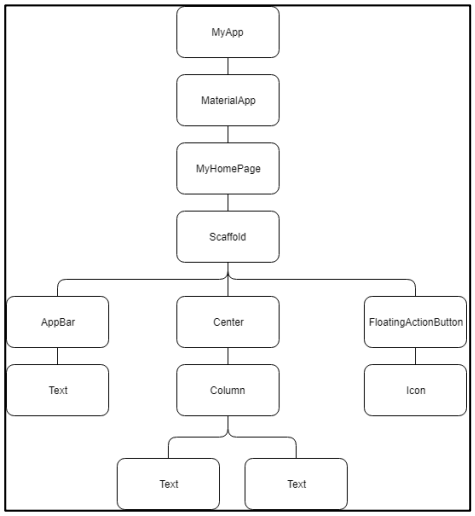
- **Animation**: tween and physics-based.
- **Painting**, **Gestures**: management of painting areas (where users can draw something) and native gestures.
- **Foundation**: it allows to create widgets in an easier way. If we want to design a widget different from the ones provided by the platform, it is possible to customize one of the standard widget interacting with the foundation layer.
- **Dart:ui**: manage communications with the Flutter engine.

Widgets are the 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, …)

Widgets can be defined as hierarchy based on composition and they allow to manage events.



Widget tree definition: the app itself is a widget, the app that uses Material UI is a widget and then there are all the other components.

Note: to align the elements, it is possible to use a grid layout called “Scattfold” and this is another widget which contains all the element (widgets) to align.

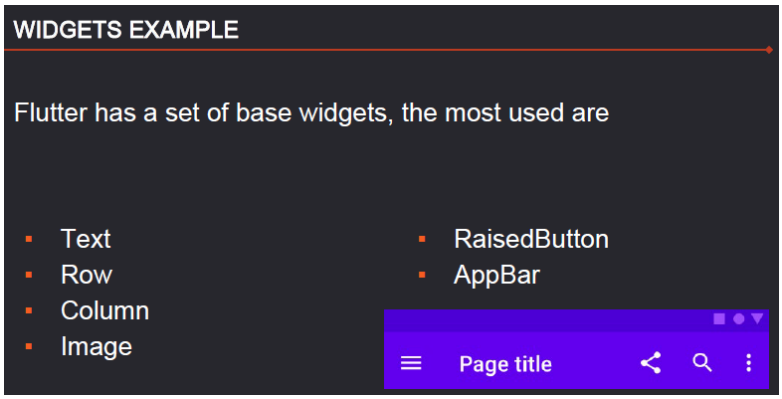
A widget is created via the *build()* method, which is called on the root of the tree and then is activated via a cascade to all other nodes.

Widgets can be:

- **Stateless**: the widget remains always unchanged (e.g. a lable).
- **Stateful**: those widget that can change → we need to save the widget state with method *setState()* (e.g. an input field).

When building, and hence loading the elements via a render, **the framework distinguish between stateful and stateless widgets**: every time an event occurs or a change is made the *build()* is not called on all elements (hot reload), since it is not necessary to rebuild stateless widgets, which cannot change. Only if the state of a stateful widget has changed the method is called, meaning we are able to save time and energy.

Flutter allows to build a widget from a particular point of the widget tree and this allow to save the state even during testing (?).

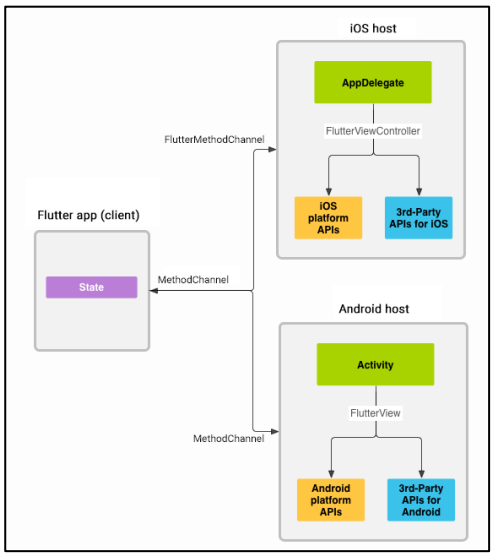


For instance, the AppBar is a mix of 6 different widget.

PLATFORM CHANNELS

Allow communication between Dart and specific code of each platform. Channel types:

- BinaryMessages: to transmit bits essentially.
- MessageChannel: to transmit messages.
- MethodChannel: to call methods from two different elements.



CODE FORKING

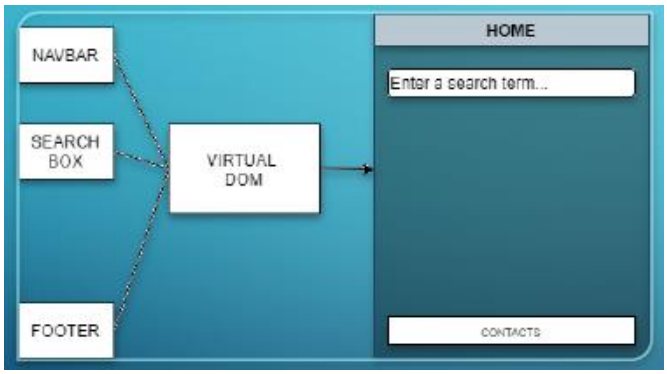
The method channel, calling a method, will make the Flutter application communicate with the iOS or Android ost, which will provide the specific APIs.

2.6 React Native

React Native isa cross-platform open-source framework that implements Facebook's ReactJS (React.js) library. It is a framework for creating user interfaces developed by Facebook.

- ⇨ **CROSS-COMPILED** approach. Since initially React Native was based on the counterpart of React we can say that it combines the advantages of web technologies with the pros of the cross-compiled approach.
- ⇨ Language: **JSX - JavaScript and XML**. It is similar to HTML, and it can be easily read by a web developer who never saw a JSX line of code.
- ⇨ It can be used for developing Android and iOS mobile applications. It ensures good performances, and it reduces the learning curve since it is based on JSX which is similar to web technologies, compared to Flutter which has its own language.
- ⇨ **Component-based**: every different part of the application is a component, and every component is independent and reusable. Components are JavaScript functions that accept input data (props) and return react elements.
- ⇨ **Uses a virtual DOM that makes it very fast**. React uses a JavaScript representation of the DOM, the VIRTUAL DOM. Initially the DOM and VIRTUAL DOM are the same, then whenever we modify something, we are modifying the VIRTUAL DOM. Then when the application run, the VIRTUAL DOM is used to see all the components that has been changed during the development, in this way it is possible to modify the DOM with just those modified components → similar to the *hot reload* for Flutter.
- ⇨ Showcase: Facebook, Outlook, Skype, Discord, Puma, etc.

Example of React Native structure:




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

Props are used to configure a component for rendering. They are read-only and influence other components with a top-down approach, meaning if the parent of an element receives a property all its children will also receive it.

```jsx
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>
      );
    }
  }
```

State is used to keep track of any component data that is expected to change over time due to a user action, network response, etc.

It can be used to influence components in a bottom-up approach (Child to Parent).

React Native makes use of the **Redux Library**, that is used to make the state update synchronous. In order to connect Redux to any application, we will need to create a **reducer** and an **action**. The latter one, the action, is an object, with a type and an optional payload, that represents the will to change the state of a component. A reducer, on the other hand, is a function that takes as an input the previous state of the component and an action, and returns as an output a new state. Also, Redux introduces two functions, mapStateToProps, and the mapDispatchToProps.

React's Native components are divided in different classes, where we can find the **community components**, created by other developers, the **core components**, installed in the framework itself, and the **native components** created by the developer itself. Thanks to a bridge, **all these components are rendered as native**, according to the device's operating system, allowing a native look&feel both on Android and iOS operating systems.

As a sum-up, we can say that the principal features of this framework are:

- A native look&feel on every major mobile operating system.
- The possibility to create and use personalized components as if they were part of the React Library.
- An active community available to give very responsive support.
- Hot reloading.
- The possibility to integrate React Native within an already existing mobile application.

...

3. Store deployment

If we have developed our application using a hybrid, interpreted or cross-compiled approach then this phase is crucial because we need to deploy the application on the different stores.

More think that store deployment step is the easier one, but it is not so true because there are many elements to consider.

Once completed the development of the app, there are two critical steps:

⇨ **Test**

⇨ **Deployment**

Both the stores, with different accuracy levels, test the applications before adding them to their offer. Therefore, it is essential to deeply test the application before the deployment phase.

Different approval times for Android and iOS: Android typically allows every application and then if there are a lot of bad reviews by users the app can be removed. iOS requires more time to publish an app because they have a more precise method to assess the app.

The deployment procedure asks for different screenshots of the application, both for the tablet and the smartphone versions.

The deployment phase should not be underestimated, as it may take some time and several temporal constraints are not manageable by the developer.

The information explained here, regarding app ranking, has been retrieved by reverse engineering since both Google and Apple do not share their algorithms that use to rank applications in their stores.

3.1 Google Play Store

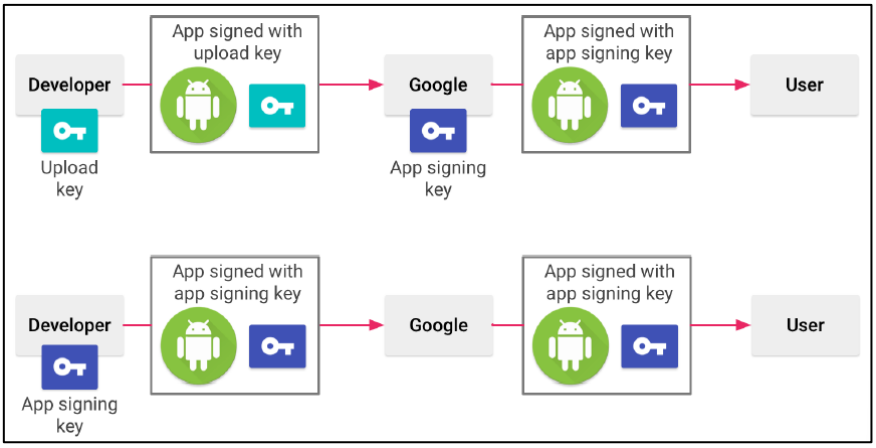
In addition to a Google account, it is necessary to have a developer account on the Google Play Developer Console:

- https://developer.android.com/distribute/console/
- 25$ unlimited duration, unlimited number of apps, it allows different accounts with different roles.
- Credit card must be available on the information of your account.

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.

APPLICATION SIGNING

Both Android and iOS require a signing process but in Android is simpler:



We see that the *key of the developer* is used to sign the application created, then successively Google creates an *app signing key* which is used to sign the application together with the developer key.

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

Where *mykeystore* is the name of the app and *aliasname* is the name of the alias. The RSA command in this example gives an indefinite validity to the key.

MANDATORY INFORMATION

- Name
- Description
- Screenshot for smartphone (at least two, min 320, max 3840, jpg or png24 bits no more than 2:1) and tablet (at least one screenshot for 7 inches tablet and one for 10 inches tablet), Icon (png32 bits (with alfa), 512x512). The more screenshots, the higher the probability of the application to be accepted in the store. They are also useful to convince the users to download our application among the ones in the store.
- Feature image for the presentation page (1024x500, jpg or png,24 bit). More images we upload more we attract potential users.

- Other images (android TV, wear OS, promotional image, etc.), but not mandatory
- APK
- Categorization (type and category)
- Content classification: necessary to determine the minimum age for the app. Requires to compile a questionnaire
- Contact email
- URL with privacy information
- List of compatible devices
- Prices and countries where it is available
- Is it suitable for children?
- Does it contain advertisements?
- Does it contain in-app products?

OTHER SERVICES

App signature can be managed using the console. It is possible to add different **app localizations**, which are not simple translations but also adaptations to different countries cultures, which can be useful to promote the application. 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)

PRICES AND DISTRIBUTION

Once an application is deployed as **free**, it is not possible to transform it into a payment app, and the best solution if this change is necessary is to create a new project.

The **price** can be automatically converted into different currency of other countries or can be manually defined (in this case, prices can be different). The **Freemium** version (free base version, advanced paid functionalities) is a good solution to let the user evaluate the app before buying it.

In any case, Google keeps 30% of the net amount.

TIMING

Once uploaded, the app is not immediately available because Google tests it. If everything goes fine, it is uploaded on the Play Store within hours or a day.

Attention: Google policy is different from Apple policy; on iOS the app can be removed if the quality level is not considered enough. Pay particular attention to app categorization and content classification: wrong information can determine its removal.

3.2 App Store

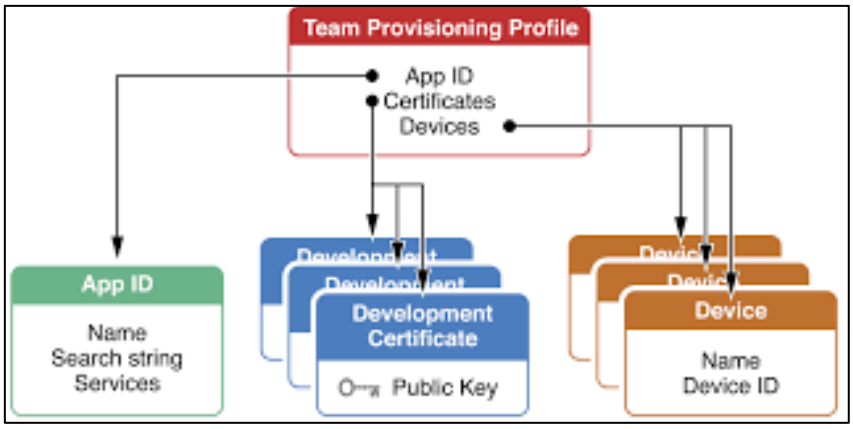
Publishing an application on the Apple Store requires an **Apple Developer Program account**.

- Cost: 99$/year
- Necessary for signing the application
- Registration as a private user or as a company

Moreover, an App Store Connect account is necessary (https://appstoreconnect.apple.com/)

The entire process is much longer than the one for the Play Store. 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 and it is also necessary for high-resolution screenshots (which are a crucial factor as we'll see).

APPLICATION SIGNING



An application must be signed with the so-called "**Team Provisioning Profile**", which contains the App ID, used to identify the app with a name, the list of all devices on which the app can be installed (iOS devices, Apple Watch, Apple TV, or Mac → a maximum of 100 devices per profile) and all the certificates used to sign the application. There are two types of provisioning profile:

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

Apple certificates are digital documents that provide a unique identity for a developer or a device. There are two types of certificates:

- **Development**: used to sign apps during the development phase. Since there are several restrictions on installing an app on Apple devices, the development certificate allows developers to install and test their apps on physical devices before distributing them to the App Store.
- **Distribution**: used to sign apps that will be distributed to end users. This certificate ensures only authorized apps can be installed on users' devices.

Signing process:

1. Developer creates an Xcode app.
2. Generate a development certificate or use an existing certificate.
3. Create a development provisioning profile that includes the certificate and devices on which the app will be tested.
4. Sign the app with the development certificate and provisioning profile.
5. Install the app on your device for testing.

App Deployment:

1. The developer creates a distribution certificate or uses an existing certificate.
2. Create a distribution provisioning profile that includes the certificate.
3. Sign the app with the distribution certificate and provisioning profile.
4. Submit the app to the App Store for review and distribution.

Signing iOS apps is a complex process that requires an in-depth understanding of the different components and their roles.

The procedure to create and generate certificates requires an **Apple Developer Program** account.

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.

Once created and installed all the certificates, it is possible to create a .ipa file (Xcode is the tool for native development). In our case, we use the Corona/Solar 2D framework that allows sending the app directly to the Apple Store without opening Xcode.

Project advice: select the University of Padova team provisioning profile, in this way the app can be installed on the selected devices, and it can run for 1 year, otherwise if we use our account the app can only run for 7 days.

NOTE: TestFlight is Apple's platform for beta testing iOS, watchOS, and tvOS apps before they are released on the App Store. It allows developers to distribute pre-release versions of their apps to a group of testers for feedback and testing purposes → a sort of "private AppStore".

MANDATORY INFORMATION

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 is:

- Platforms
- Name. The name is much important because it can improve the app position in the results page. For instance, if we are a big company (as Nike) we should put the name of the company inside the app name → in this way users may easily find our app.

- Primary language. Other languages can be translated by buying a specific translation service.
- Packet ID (is the ID chosen when creating the certificate)
- SKU (a unique id, not visible to Apple, defined by you)

Once created a record on iTunes Connect, other pieces of information are required (Panel «App information»):

- URL with privacy information
- Subtitle (optional)
- Category and classification: attention, very important. It is possible to choose two categories and one or two subcategories.
- License agreement
- Screenshot

It is possible to save a partial draft and continue later. The app can be: free; for payment; with in-app payments (freemium).

The App Store requires to add at least one screenshot for each supported device, because resolution can deeply change.

- 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).
- Do not underestimate this step!

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

- Application description and keywords: keywords cannot be modified
- URL for support and customer care information
- The build (or binary)
- Icon (1024x1024 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).

VERIFICATION AND TIMING

Apple checks that your app:

- ✓ does not contain malware or not authorized content;
- ✓ has and high-quality standard of the interface and follows the indications;
- ✓ works!

Tests are made manually, in the past they required on average two weeks, actually one week (if everything goes well the first time).

App Store Connect allows monitoring the app after release, providing information about

- downloads and sells.
- crashes (only if users provide consensus).

3.3 After the deployment

After the app deployment an important concept to remark is: "**do not abandon the user!**"

Especially for Android applications, it is crucial to monitor app usage:

- With "Abnormal stops and ANR", "Application Not Responding", it is possible to see malfunctions and with which devices, and error reports from the users.
- Crashes are listed only if the user accepts to send the report.

Publish new versions both to solve bugs and to provide new functionalities.

App optimization for the stores: the higher is the position of the app on the listing, the higher probability of new installations.

APP STORE OPTIMIZATION

Stores are huge. 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 all the other information 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 search engines, both Google and Apple do not publish ranking criteria.

⇨ Onsite criteria that influence ranking are:

- Keyword, both inside the name and between ones defined when the app was inserted in the store
- Presence and type of icon
- 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
- Number of removals
- Number and average of the evaluations (even negative evaluations can improve the ranking because they increase the total number of reviews).
- Ratings evolution

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

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 in 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, that 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.

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.

The description is a maximum of 4000 characters, but it 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

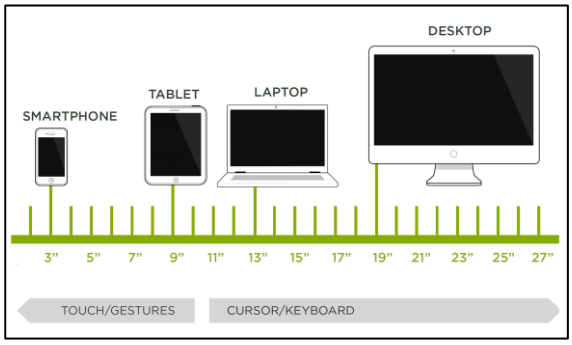
- Ratings are important even if not positive.
- Customer service helps improve ratings.

4. Mobile Design

User interface design for mobile applications must consider several aspects. The main differences with desktop counterpart are:

- Device size
- Computational capabilities
- Operating system (real time: I can suspend an app and do something else; a desktop device schedules its tasks and complete them)
- Interactions! (Touchscreen, Sensors = movements, Vocal input/Output, More direct/natural interaction)

Touchscreens are available in several different situations: ATM, machines for electronic sell, Informative panels (ex: museums), Tablet/smartphone, Mobile computer.



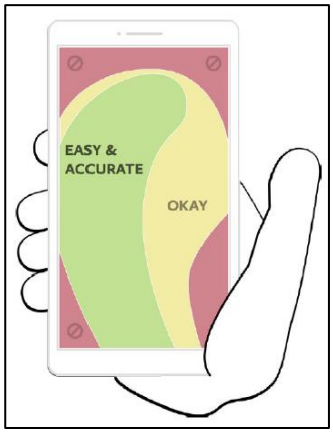
Other notes:

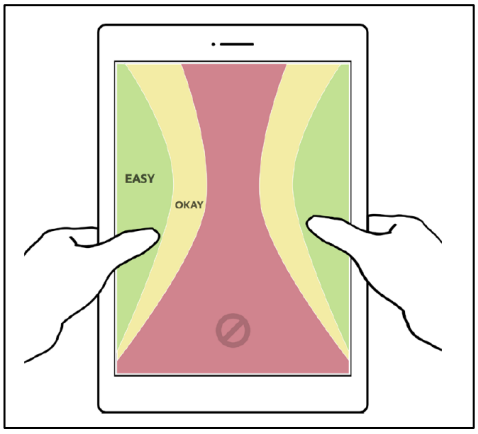
- When the portion of touch interface becomes bigger, it becomes harder to manage, we will see why.
- Using a tablet instead of a smartphone is different: tablets are usually used by both hands and typically when we are sit somewhere; the smartphone can be used everywhere, another aspect to consider!
- Vertical interfaces are not so effective as we think, **if the interface requires a lot of actions users may get tired** and we need to avoid it; this is knows as "*gorilla arm problem*" (e.g. think about the McDonald's touchscreens used to order food, their implementation is effective just because the interaction is fast and require few steps).
  "*Touch interfaces don't want to be vertical, it gives great demo, but after a short period of time, your arm wants to fall off.*" (Steve Jobs)

4.1 Design a good interface

The reachability of different components of the interface depends on how we hold the device (e.g. one finger, two finger, working with one/two fingers). The main usage is keeping the phone with a hand and use the thumb to interact with the app.

SMARTPHONE & TABLET COMFORT ZONES

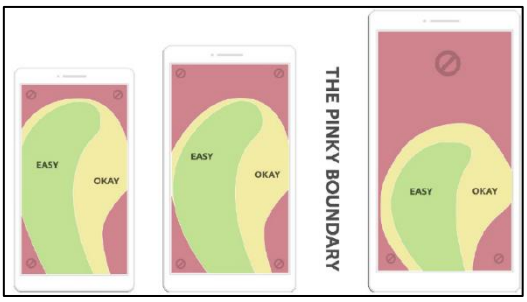




There are different zones of the user interfaces to consider, some of them easily reachable, other less (note: there is no problem for left-hand usage, the zones remain the same).

The green zones are the so called “easy & accurate zones”, it is the easiest and more accurate zone where most used widgets should be placed in order to be easily reachable by users. Then there are the yellow zones (“ok zone”) where it is still possible to reach widgets, even if in a less precise way. At the end, there are the “red zones”, the most dangerous ones because they are the hardest zones to reach with fingers.

So, which widgets for which zone?



Consider even the size of the smartphone, as the size increases the reachable zones become smaller → for this it is important to group controls, in this way the user doesn’t get tired.

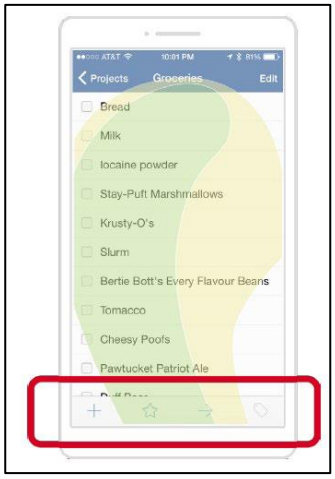
With the increasing size of the screen, users tend to use the device with two hands for a better holding. Increasing device dimension means increasing weight. 88% of tablets usage occurs while seating, against 19% of smartphones. Tablets are used on a holding surface two out of three times. Big devices are used in a similar way to laptops but

- Mouse is usually moved easily (you can move from the left side to the right one moving just a little bit the mouse), whereas fingers are moved by the hand and require a higher effort.
- It is essential to group controls together to avoid user’s tiredness.

THUMB RULE: identify the most frequently used controls/widget and put them in the comfort zone (*thumb zone*). It is also very crucial what to put outside this zone: for instance, controls for data modification to avoid unwanted edits or data loss (e.g. login [inside] vs logout [outside]).

CONTENT ALWAYS ON TOP RULE

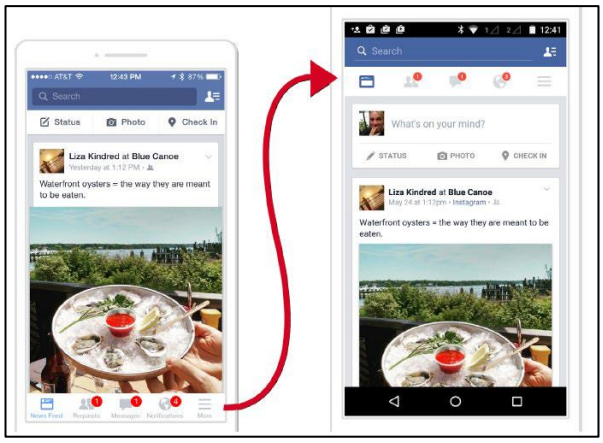
With classic interfaces, the websites ones, priority is given to interaction elements, like buttons clickable by a mouse (in desktop interfaces the controls in fact are usually on the top area of the device, so the first elements users see). Touch interfaces change the interaction tool: the mouse cursor is tiny, while hands are bigger and less precise. Furthermore, consider that **hands can hide part of the interface, so it is important to calculate their encumbrance**; important data must remain visible, if necessary, they can hide the controls → example: letters shown when writing with a digital keyboard on the smartphone; password partially shown when writing it, then the stars replace the digited character → in this way users always know what they have digited/click.



The "Content always on top" rule forces to leave the content in the center and move controls above, below, or on the sides.

A problem for this implementation is related to the fact that many devices, typically Android ones, have their own OS buttons (another control bar) already on the bottom and moving the control of your app on the bottom can lead to some problems → for some users may be hard to understand which are the controls of the app and which the OS ones; so, in general we should avoid to use stacked bars and that's why we need to separate them.

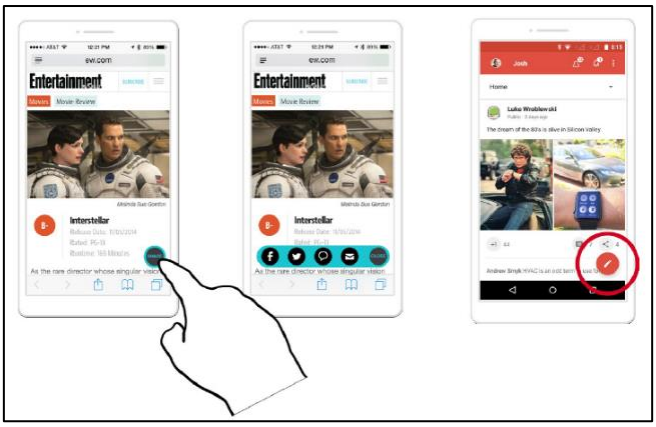
For applications like Instagram, where frequent data modifications may occur, the controls must be inside the comfort zone.



Considering the Facebook study case on Android, moving the controls on the top, outside the comfort zone can be counterintuitive but in fact this has been done for the already OS buttons mentioned before and even because the types of interactions are different → in fact those functionalities do not interact with the displayed content, for sure they cover the content but in this case is acceptable.

Since there are different mobile OS with different UI configurations, main rules can be summarized in this way:

- **Android**: controls must be on the upper side of the screen.
- **iOS**: controls must be on the lower side of the screen.
- **Phablet** (touch screen devices with screens having a diagonal of 6 to 7 inches; above 7 inches, however, we can talk about real tablets): controls must be on the lower side of the screen**.** It is possible to introduce a **floating trigger button for frequent operations.** Swipe usage, especially for the tabbed layout.

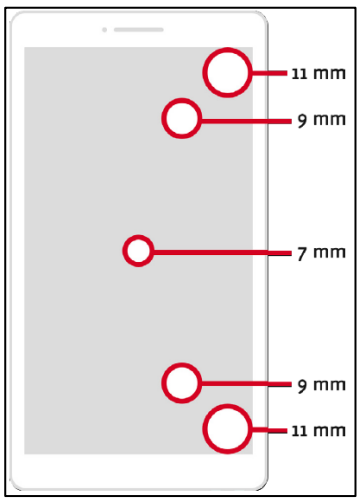


FOCUS ON LAYOUT FOR TABLET: with bigger screens, it is difficult to have a unique overview, but designer must consider that **eyes move from top to bottom**. Buttons for interaction must be at the top or on the sides of the device. Prefer corners at the top and not the center position because otherwise users cover the content. 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 effort is required during interaction**. It is crucial to reduce the number of interactions → group together interaction elements. A typical error is saying "I have a bigger screen so I can put more widgets".

HOVER PROBLEM: with except to several proprietary solutions (ex: 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. This can work but we are requiring two taps for every interaction! If we really need to have the hover this can be a solution, otherwise apps should be designed without considering hover event.

According to a Google study of 2013, 83% of websites provide interaction buttons too small to be used with fingers. **How big is a human thumb?** The minimum is 8 millimetres for a child; the maximum is 18 millimetres of an adult; **the average is around 11 millimetres**. The minimum size to use is 7 millimetres, which can be increased to 9 mm with big tablets.



Size vs position

The precision depends also on the position of the widget in the screen; for instance, if the widget is inside the comfort zone the minimum size (7mm) should be enough, otherwise I should increase the dimensions following the left scheme.

The dimension of the thumb is in millimetres, but this unit of measurement is not the best one for interfaces design. The following tables gives appropriate units of measurement and recommended values:

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

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 millimetres must be at least 2 millimetres away.

⇨ It is a good idea to **not crowd interfaces**.

JUST-IN-TIME INTERFACES (What should I put in the interface?)

A good interface must provide only what is necessary at that moment, what is called "Just-In-Time interface". 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. ➔ Remove everything is not necessary at a specific moment.

In this example, the first screen provides information for more locations, then if we are interested in a particular one, we can tap on it and see more information (second screen).

Try to put all the information together!

With increasing bandwidth available, and the availability of a local database, the number of taps is less critical:

- Distinguish between useless and quality taps. A **quality tap** is a tap that adds new information, completes a task, or simply adds a smile (emotional design). 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.

THE PROBLEM OF LONG PAGES

A big issue in mobile design is related to long pages, both along vertical and horizontal side. 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. They cause a loss of overall vision, they make difficult to grasp the connection between different objects, since users do not understand what comes before or after. 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**. A good example of this better approach is the application Zalando Prive which shows to users items that can be bought without manually searching for them.

A question could then be when the carousels are effectively useful. Some examples could be:

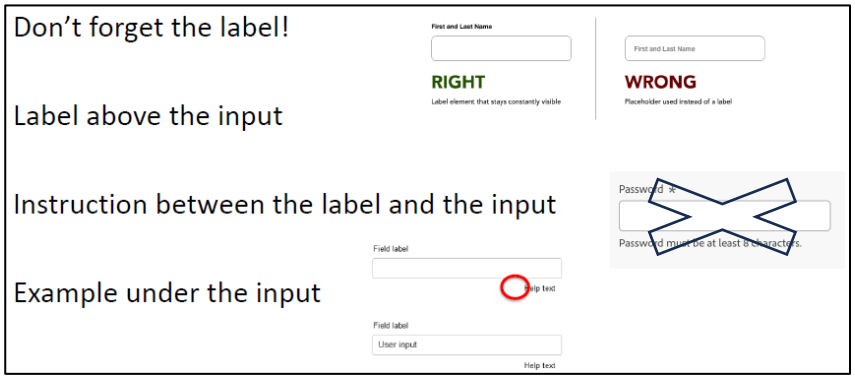
- Linear data: user knows what to expect (ex: weather).
- Random browsing on interesting items for the user: pictures, slide show. They work better if the user knows what to expect, for example, in a known context (http://shouldiuseacarousel.com/).
- To break up very long forms: in this case, they cannot advance automatically.

THE PROBLEM OF LONG FORMS

21% of e-commerce, that 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**: a study shows that for a contact form, 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. The lack of the tab control is related to the lack of the keyboard and all of the functionalities it provides. To help users we should:

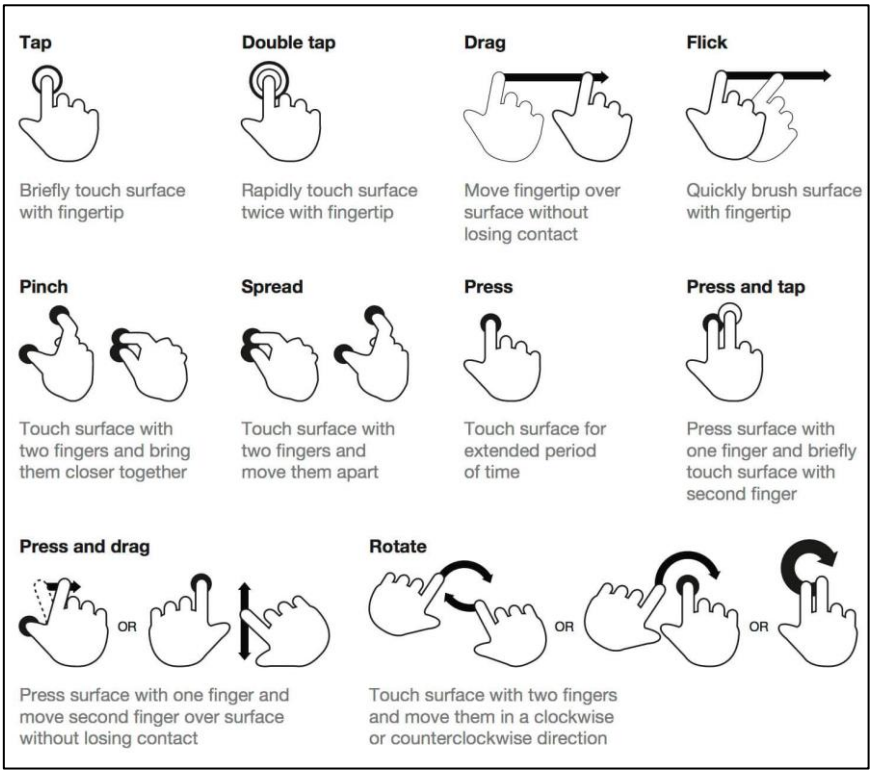
- Provide the correct keyboard for a specific input.
- Prefer a list of buttons to a menu if this one is short.
- Avoid too long drop-down menu (a dataset is better).
- For numbers insertion, it is preferrable to show an average value with +/-buttons if the real value will not be far away from the average.



The previous examples are not wrong in a web environment, but they are wrong in the mobile world because they are simply translated from websites to mobile devices without taking care of user habits (e.g., finger over the instructions under the label).

Confirmation dialogs were introduced to let the user think about the answer, but today do not work anymore and slow down the user (good for laptop but no more for mobile devices). It is better to **use specific gestures** (e.g., a swipe to answer calls or to unlock the device). **This movement is sufficiently difficult to be only intentional while remaining easy enough to be fast**. Also, makes possible to undo what did.

4.2 Gestures



Some definitions:

- **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 MS 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 behaviour, usually opens a contextual menu.
    - only expert users use it → this means that we need to teach the movement.
- **Long press + drag**: equals to drag&drop.

- **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 (ex. Select an element and apply an action).

Before implementing a button, we should ask ourselves if it is really necessary. In fact, buttons must not cover the content and the only presence of command controls entails a dedicated portion of the screen. The main difference between a gesture and a button is that gestures do not provide entry points, users who do not already know the movement and where to do it can encounter difficulties.

Gestures improve interface accessibility because they tolerate less precision. This is important in different situations, for instance: elderly and children usage; when the user cannot pay close attention to the interface; situations where it is necessary a fast interaction with no errors → if the user knows the gestures, there is no need to watch the screen. **Big gestures tend to become reflexes**; in fact, traditional interfaces are based on visual memory while touch interfaces use muscle memory (think about muscle memory used when playing guitar).

In the real world, buttons are used to control something not reachable (ex: fan). With touch interfaces, everything should be reachable, so before using them, always check if there is another way to manipulate the content directly.

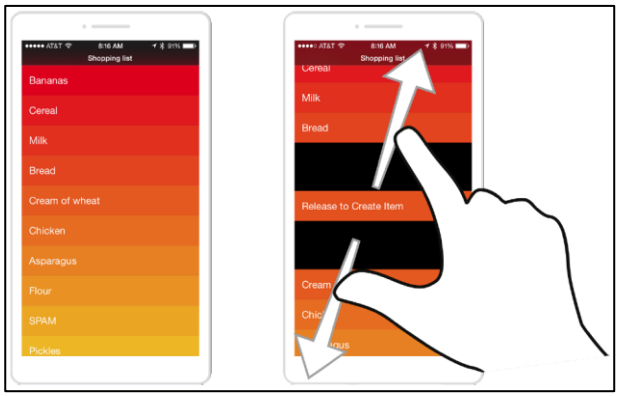
- Consider the content as an existing physical object.
- Pay attention to external conventions (ex. salt and pepper, pictures gallery).

Example: *Angry Birds* uses natural and intuitive gestures and not buttons.

The world of gestures is deeply linked to the one of metaphors. For example, a **card metaphor** is something frequently used and deeply understood by users: think of applications to manage business cards, plane tickets, or coupons. Cards suggest to us several interactions, like flipping them, or putting them in stacks. But, in the mobile world, we flip a card by clicking on a button, which is not really manual: this teaches us that **is always important to help the user understand and get natural with an interface**.

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 (e.g. drawing apps). Anyway, even if we follow the real word, everything must follow conventions → do not betray user expectations, the context is important.



This example shows an application for a list of groceries. 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. **It has to be taught to the user since it is not directly intuitive**, but once learned it becomes very natural.

Even if it is important to use natural gestures, **it is possible to introduce complex gestures** that can be used as the keyboard shortcuts, for instance swipe to delete something instead of tap on the bin icon. In this case, the design must prefer more natural but longer gesture. It is even possible to use gestures with more fingers (e.g. Display space, Multitouch support is not always optimal, 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 and the override should be avoided: with Android always start from the sides, with iOS can be completely inside the space dedicated to the app.

Circle menus are frequently a good way to reduce conflicts between gestures, they are easy to learn because they use muscle memory, and fast to use. These kinds of menu are frequently used with games, useful for primary navigation, context menu, or tools.

Disadvantages:

- Require more precision.
- Not scalable (on phones, 3 or 4 options).
- Not easy to use the first time.
- Cannot change over time, users should re-learn them!

HOW TO TEACH GESTURES?

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 that usually is not reading the instructions, he simply wants to do something. Some gestures are trivial, but often after having discovered them.

⇨ Solution: **just-in-time education**, we do not send the information at the beginning, but we wait until it is needed.

Together with just-in-time education, it is also important to consider the idea of **skeumorphic design**, which entails representing elements of our application with real-world elements. The most famous example is the cycle bin as the trash icon in a personal computer. Although skeumorphic design isn't always suited for gestures teaching, the main concept that is

important to remember is the one of the metaphor: **we need to find the right metaphor to represent our application and not betray it, in order to not confuse the users**.

A wrong usage of skeumorphic design is trying to implement too much realistic interfaces (Apple's contact book); in this way users expect to use the app in the common natural way they adopt with the real counterpart and when this does not happen the metaphor is broken as well as users' expectations. Making interfaces too much real can be dangerous, because usually the following equation stands: "*Looks like ⇒ Acts like*". If this does not happen, the user will be confused. Differently, if an object *acts like*, it does not mean that it must *look like*.

There are even **linguistic metaphors** which create a relationship between tow ideas through words (often names give to features).
Example of Twitter: the term "tweet" refers to a short message or status update posted by a user on Twitter. The choice of the word "tweet" evokes the image of birds chirping or tweeting, reflecting the brevity and informality of the platform's communication style.
Example of save icon: visual metaphor become a linguistic metaphor → we do not more use floppy disks, but the concept can be used to represent the save action.

Another approach to teach gestures is to use **instructions**. It is not true that if the user needs instructions, the designer has failed to design the interface. Instructions entails a better learning when doing and teaching gradually the gestures during the interaction, not before.
Videogames solve this problem very well, the user usually does not know what to do but learns while playing through three mechanisms:

- **Coaching**: tell someone what to do is not as effective as showing what to do. Coaching provides easy instructions when the problem shows up for the first time. The key point is to **understand if and when the skill has been learned**, from that point instructions are useless: **ask the user to make a gesture + ask several repetitions of the gesture**.
- **Leveling up**: involves progressively introducing more advanced or complex gestures as users become more familiar with the app and gain experience. The leveling up 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** (see the videogame example here below).
    - Provide the necessary time to learn.
    - Example of Instagram: at the beginning we can tap on the heart icon to add a "like" to a post, then, after some time, it is possible to learn that we can do the same operation with a double tap on the photo.

- **<u>Power-up</u>**: 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. **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.

<u>MISTAKE AS RESOURCES</u>: users' mistakes help to understand what users have not learned and so that instructions are still necessary. If the user stops during the interaction, this could be a warning: An animation can help the user to complete the interaction. The best interfaces record when the user interacts, do not interact and the learning process, adapting hints and suggestions to the latter.

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.

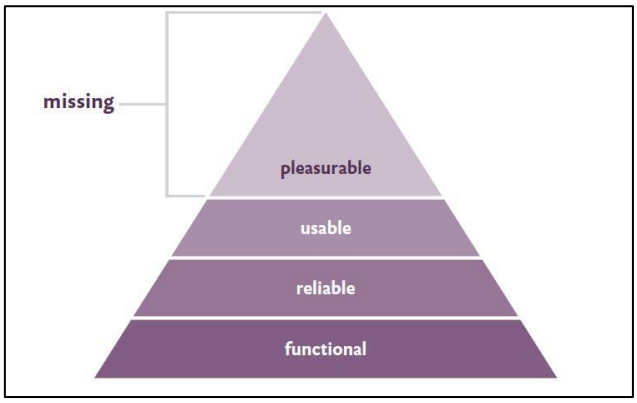
4.3 Other type of interactions

Mobile devices also allow other types of interactions besides touch:

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

4.4 Emotional design

Emotional design UI is strongly connected to the Maslow's pyramid, in order to be happy we need to fulfil all the levels of the pyramid. We can remap the standard Maslow's pyramid with the users' needs:



Users' needs:

- **Functional**: user must be able to complete the assigned task. Don't stop here!
- **Reliable**: the system must work, failures of every kind are unwelcomed.
- **Usable**: it must be easy for the user to learn how to use the system and its functionalities.
- User experience must be **pleasant**. Users should be happy when they use the app.

Amygdala is the oldest component of our brain whose job is to answer to our needs such as hungry or thirsty. The stimulation of this part of the brain, make easier for us to make our application more acceptable. **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. It is not easy understand what the users like, which is the best picture to put in my app? There is one common rule that developers should follow: **golden ratio** rule. The golden ratio is a

mathematical proportion (roughly 1:1.618) found in nature and considered aesthetically pleasing. It suggests dividing a space into two parts, where the smaller part is to the larger part as the larger part is to the whole. We can use it in mobile design, for instance, dealing with:

- Screen division: You can use the golden ratio to divide your mobile screen into sections. A larger section could hold your main content, while a smaller section could house navigation or secondary information.
- Element placement: The golden ratio can guide the placement of buttons, icons, and other UI elements. By positioning them along imaginary lines based on the golden ratio, you can create a more visually harmonious composition.

The Wolfwsvagen Maggiolino helps us introduce another topic in emotional design: its enormous success show that users are attracted to objects that are humanized. **Humanization**, 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, in this example, using really big faces is a factor in the strategy of marketing: newborn 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. This, linked with the facts that humans protect the human children, and say yes to them, tend create a positive effect in users.

Another technique that exploits emotions is the usage of **personalities**: imagine the brand as a person with a personality, and imagine how this person could speak, answer, act, etc.

Which emotions can we use? There are no rules, but we can use different emotions depending on the context. Generally speaking, the most effective are: Surprise, Pleasure, Preview, Status/Exclusivity, Reward → Never force the user to change!

A good example of exploiting users emotion is done by 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.

FOLLOW THE INSTINCT: when users must decide something, they consider the pros and cons. When it is not possible to measure everything accurately, instinct prevails. Major obstacles: Laziness, Skepticism. A good design or the use of games/incentives can help for a good decision.

WHAT IF IT DOESN'T WORK?

- Is the persona created for the brand correct? While using a bank application 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 user need satisfied? For example, in social networks, if we are able to convince multiple user 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, servises?

4.5 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 later.
- Theme: how ideas are understood when looked at as a group.

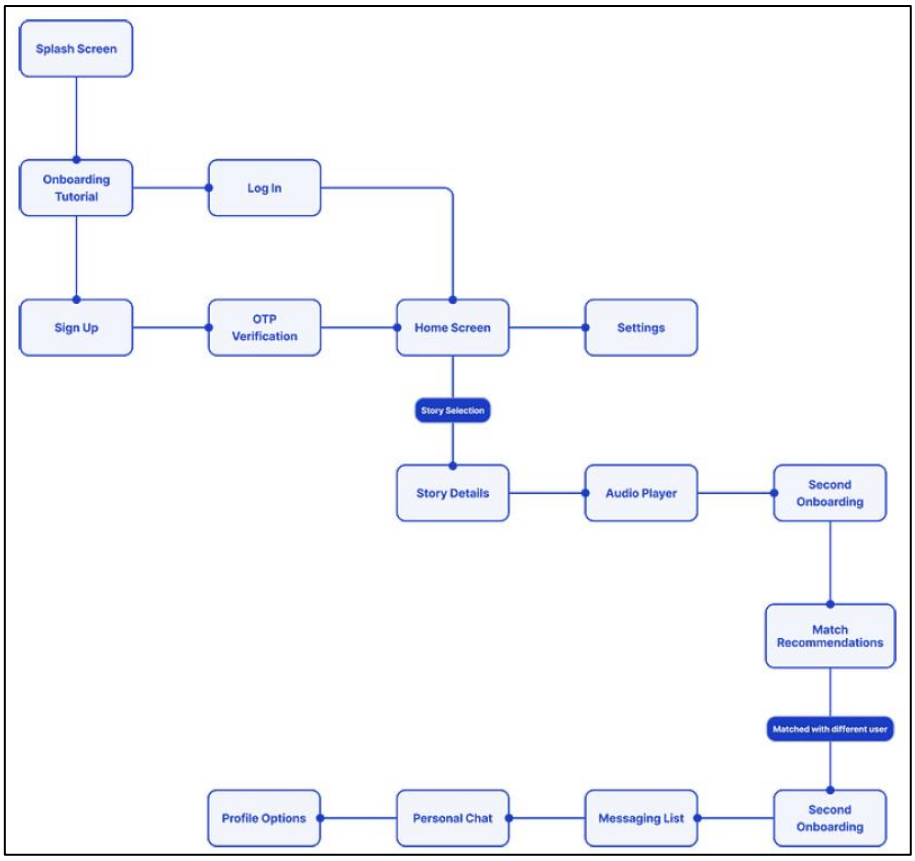
If you consider the design of an app as a storytelling, this isa good way to discover interactions.

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

How to find pieces of your app story: It's a process of 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…

To finish: **don't underestimate 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. 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.

The icon should always remember the functionality of the application. The Instagram's icon, for instance, remember the camera one, meaning that the main functionality of Instagram is image management.

B. Multimedia Data

1 Introduction

The world in which we are used to live nowadays is a **hypermedia**, which is a conjunction between ***hypertext*** (the structure we are used to associate to the Web) and the use of ***multimedia data*** (ex: play a video → video + audio). With hypermedia, single nodes contain continuous media and then we have anchors that connects different nodes, this is what hypertext do on the web (connecting information). This means that a high dimensional scenario has to deal with **difficulties related to the synchronization of different continuous media** (video, audio, animations, and so on → ex: "lip synchronization", used in movies to synchronize the audio and the video).

But what actually is multimedia?

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 (e.g., text, graphics, recorded or synthesized audio, animations, fixed or motion images).

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, …)
- Non-computer based multimedia presentations have different sources of media (ex: 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 (e.g., how can we highlight a link in a video or audio file?).
- Information humanization.
- Temporal dimension.
- Inappropriateness of permanent copies (e.g. "I cannot print a video").

Problem of representing something that is not evolving during time, such as the last frame for video (after pause). For the video there are some solutions but for the audio there is nothing to do.

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

- Methodology: behaviour models of information access.
- Technology: development and support tools, standards.
- Dramatization: emotional interpretation of information.
- Psychology: efficacy, privacy, control.

SOME PROBLEMS:

- 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.

As soon as the multimedia complexity increases the compression techniques become more challenging, but compression is essential because, as said before, multimedia dimensions are huge. Remember, the problem is reproduce, synchronize and reproduce again huge amounts of data.

We will apply multimedia strictly to Web scenario, which means we need to clarify the differences between WWW and Internet. The World Wide Web is a distributed application (an application container) using Internet as the technological infrastructure. Internet provides communication services, protocols, remote storage systems, etc. while WWW provides the user interface, the environment for distributed applications and the applications. Multimedia systems cover both worlds, but every world has its own space and sometimes boundaries are not clearly defined.

Multimedia technologies are widely independent from the Web (Information representation, Information compression, Information transmission). Web applications are not the most critical, and do not require the highest resources (e.g., video on demand, multichannel communication). Web evolution is imposing new goals and so new quality standards.

2 Media classification

Media can be classified depending on:

- Content: text, image, audio, video.
- Media dynamic: diffusion and media usage, protocols and network technologies.
    - Static media: images and text, so information that do not change over time.
    - Dynamic media: audio and video, so information that change over time.
    - Temporized media: live streaming, so video that can only be seen in a particular moment, meaning information changes over time in function of temporal constraints.

All media (text, image, audio, video) 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 between those three properties, meaning usually if we are able to have a low encoding weight we will have lower quality.

Not considering advantages and disadvantages, we have goals on what a multimedia encoding algorithm should do:

- ✓ reduce the size of persistent data;
- ✓ optimize the transmission time over reproduction time for continuous data (videos on YouTube!);
- ✓ fast decoding for data encoded on conventional and not conventional architectures. We can use more time to compress data (upload a video) but the time required to decompress data (play a video) must be as short as possible;
- ✓ quality control of decoded data;
- ✓ provide support to replace missing data. We cannot simply retransmit data because otherwise users may wait for a long the desired resource. One possible solution is the repetition of data, in this way if we lose a packet there is another completely equal to the previous but on the other hand the size of the object becomes bigger → again we need to find a compromise between size, time and quality.

STATIC MEDIA

User reads information with his/her own timing. Data transmission does not impose temporal deadlines (but with reasonable limits) → the time users take to get the information (read a line) allow systems to upload “slowly” the resource. Persistent information. Users can read the information several times → Ex: text, images.

Data size is not a problem. A higher size means a higher compression time, but this is not a problem for the user, which has to interact with it. In most cases, information diffusion and usage of static media take place in separate moments: the user can download and display

information separately or save them on the local storage of their devices, or, again, cache them in their browsers.

DYNAMIC MEDIA

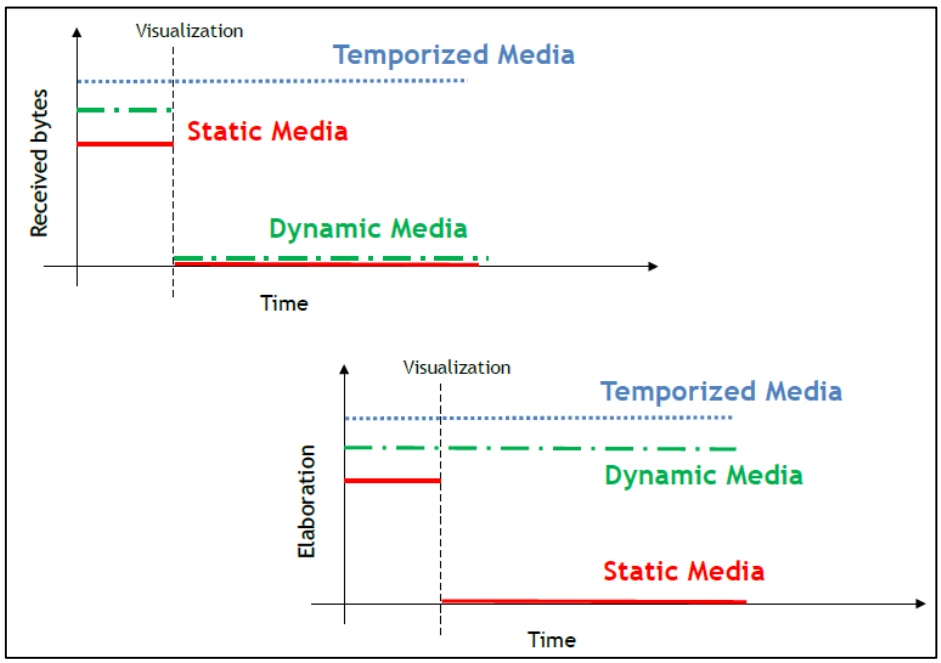
User must follow information evolution in real-time. Diffusion can be delayed, but information meaning can change (ex. games → "if you don't give the answer in a certain amount of time you can lose points"). Information is not persistent (sometimes can be reproduced several times) → Ex: animated presentations, YouTube, video files.

Differently from static media, here size is a problem, since the information is frequently big, and should be used in a predefined time, like during a video. 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.

TEMPORIZED MEDIA

User must access the information following the timing schedule of the information. Diffusion must follow media isochronous timing (must be reproduced at a defined fixed frequency to be understood). Information is not persistent. The data cannot be seen twice → Ex. audio and real-time video. The size of the information in this case is proportional to the reproduction time.

STATIC VS DYNAMIC VS TEMPORIZED MEDIA



3 Information Compression

Multimedia data size is too big to efficiently transmit and store data without using data compression. The main goal of data compression algorithms is to **reduce storage hence reducing transmission time**. Decompression is necessary and needs to be in real-time. Data compression can be in real-time or deferred time. Fortunately, it is 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 (or represent) the same information (**entropic compression**).
- The reduction of the information to memorize or transmit (**differential compression**, semantic compression). It does not cover all the media, the algorithm understands which are the data that I can lose and which not.

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:

1. **Transformation**: information data (ex: pixel of an image) are transformed in a new domain that requires less bits to represent data values.
2. **Quantization** (only in lossy compression): obtained values are (can be) grouped in a smaller number of classes and with a more uniform distribution.
3. **Encoding**: information is encoded based on the new classes and values (domains), together with the encoding table.

Lossless compression techniques (used for all types of data):

- **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).

Lossless compression seems to be the most incentive one, but it is not always efficient.

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.

3.1 RLE

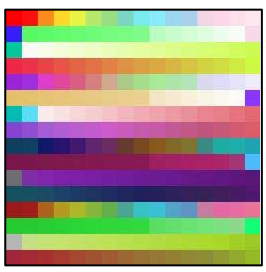
When data contains a lot of consecutive repetitive values, there is a lot of redundancy that can be deleted. Long sequences of the same value can be substituted with a **repetition symbol**, **the value**, and **the number of repetitions**. Compression is efficient for sequences of at least three equal values. Suitable for images with little details (ex: a uniform background). RLE is implemented in the bit-map BMP format.

Initial sol: BGFDDDDDDDDDIJUPPPPPHYTGBUYYYYINNNNNNGHHHHHHHHKPPPPPPP
→ BGF**@9D**IJU**@5P**HYTGBU**@4Y**I**@5N**G**@7H**K**@7P**

This solution is limited to the size of the domain on which we are working, in fact we need to choose a symbol to represent repetitions that is not used in the data representation. In the previous example "@" is problematic if we are working on an email's domain.

More general solution → a repetition requires two bytes: one starting with bit "1" and the number of repetitions (7 bits), and one for the repeated character (8 bits). If the character has no repetitions, its byte begins with the bit "0" and then the remaining 7 bits represent the character. Following this reasoning, RLE is not suitable for sequences with less than 3 repeated characters, since just 1 character means 1 byte (standard scenario → no gain), 2 requires 2 bytes (as the RLE → no gain) and only with 3 we have an effective encoding.

RLE works well with several repetitions, otherwise RLE increases the original file size. RLE is suitable for artificial images or with few details. On the other hand, it is not suitable for text and images with several shades of color or high level of contrast.



Enlarged image of a 16x16 pixel file, with 256 unique colors.

This file, saved in BMP not compressed, is 822 bytes. Saved instead in BMP format, using LRE algorithm, is 1400 byte, 1,7 times more the original size.

3.2 Shannon-Fano algorithm and Huffman Code

The **entropy** of a source of information is the measure of the quantity of information in the bits flow that has to be compressed = number of bits (theoretical) necessary to encode.

Note: "theoretical" because the formula can return rational numbers as result and if the result is 2.1 bits it is clear that we need to use 3 bits.

$$H(S) = \eta = \sum_i p_i \log_2 (1/p_i)$$

- p_i is the probability of the i^{th} element
- $\log_2 1/p_i$ is the minimum number of bits necessary to encode the i^{th} value so that it can be recognized between other elements

Based on the probability of finding a specific value (or symbol) inside a sequence, we can create an entropy encoding which **encodes a value with a different number of bits based on its frequency** (probability of appearance).

⇨ Elements that appear more frequently in the message have a higher probability (p_i) and therefore contribute less to the overall entropy. These elements can be encoded with fewer bits because they are predictable.

⇨ Elements that appear less frequently have a lower probability and contribute more to the overall entropy. These elements require more bits to encode because they are less predictable.

From Shannon's Theorem, for any lossless compression scheme, the lower bound of the average code length *L* (compression scheme) is the entropy of the information source *H(S)*:

$$H(S) \leq L$$

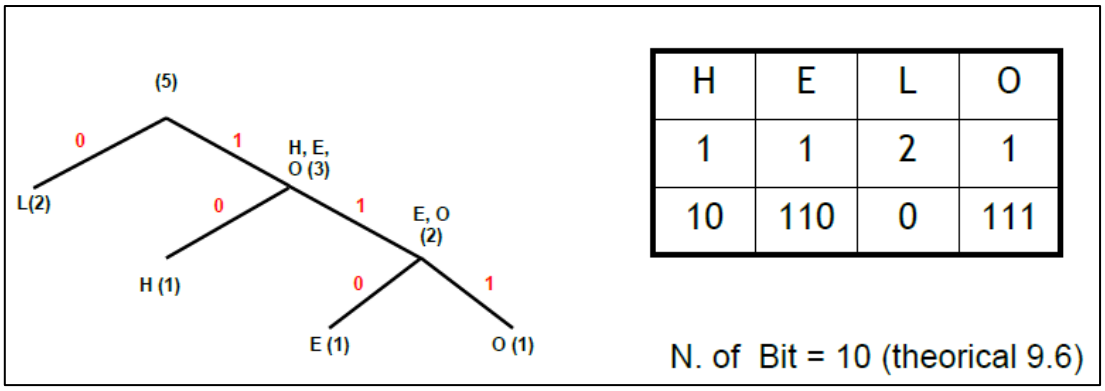
In other words, **no lossless compression scheme can achieve an average code length shorter than the entropy of the information source**. If *L* was lower than *H(S)*, it would imply that the compression scheme is compressing the data beyond its theoretical limit, potentially losing information in the process.

Shannon-Fano Algorithm → It creates a binary tree with a top-down method. Each node is a symbol or a group of symbols:

- Sorts nodes based on the number of occurrences.
- Recursively divides the set of nodes into two parts, each containing, approximatively, the same number of occurrences, until each part has only one symbol.

Note that typically characters are encoded using ASCII code with 8 bits per character, with compression techniques we want to reduce the number of used bits per character (or symbol).

top-down



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

N. of Bit = 10 (theorical 9.6)

Note: count the number of occurrences for a symbol, order the list and then cut the list, then the list is not ordered anymore! The recursive process just focuses on cutting the list in a way we have two balanced lists.

The first step is dividing L, which appears twice, from H, E and O (three). After that, we can arbitrarily divide these letters since they each appear one time. In the tree, each right branch gets the label 1, while the left 0. The labels are assigned based on the path that need to be followed in order to find a letter in the tree. The total number of bits is calculated in the following way: # of symbol occurrences * number of bits used to represent that symbol → so, in the above example the total calculation is the following: N = 1*2 (H) + 1*3 (E) + 2*1 (L) + 1*3 (O) = 10 bits.

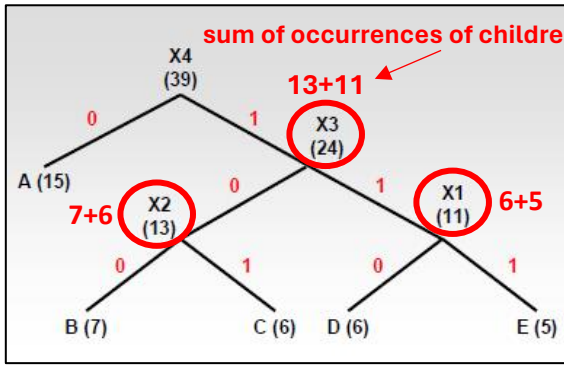
Observations:

- Here the ASCII code would have used 40 bits, so the compression result is efficient.
- We are near the optimal solution which is the entropy: 9.6 (theoretical) vs 10.
- Any other symbol representation, calculated by the algorithm, cannot start with other symbols content (e.g. encoding of H = 10 so other symbols cannot start with "10").

Huffman Code → 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 number of occurrences, creates a subtree with occurrences equals to the sum of the occurrences 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.

bottom-up



simbolo	# repetitions	$\log_2(1/p_i)$	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)

Note: after creating the first tree (the one in the bottom right corner) we need to select the two nodes with the lowest number of occurrences, selecting from: B, C, **X1**.

The final number of bits that are needed to encode a given string can be represented as a sum of two quantities: the number of total bits needed for a single character, which can be seen as the number of bits used for the code, times the number of occurrences of the symbol (so, for A it is 1*15, B is 3*7 and so on), plus the bits needed to represent the table. The best way to represent the table requires 8 bits (needed to store a char) times the number of different symbols, in the example 8*5, plus the bits needed for the encoded sequence length which are again 5 so again 8*5. Finally, the above example will require 87 + 80, so 167 bits. At the end we need to add two columns. Note that to save the table only the second and the fourth column of the above Huffman table are crucial: I need to represent all possible symbols and the length of each symbol encoding. 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.

Huffman and Shannon algorithm may provide the same results, but Huffman is optimal, so it is more probable that Huffman will provide a solution really close to the entropy of the information to represent. Anyway, both algorithms have a problem: the drawback of these methods is 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. The complexity of the table increases as the number of symbols, used to represent the information, increases (look for Huffman vs Shannon comparison in the slide).

3.3 LZW

It dynamically builds the vocabulary of the symbols using fixed codes for sequences with variable length

```
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;
```

	INPUT		DICTIONARY	
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		

LZW performs better than Huffman because LZW is not based on the entropy theory since it associates a fixed code to a length variable string of information. Huffman is the optimal but just between entropy algorithms! **LZW does not put the dictionary directly in the file**, as Shannon-Fano and Huffman do, and this is the key of the LZW algorithm: the dictionary is dynamically created. LZW is not so efficient at the beginning, compared to the previous algorithms → the first time I see a character I don't encode it, but I will just read it. 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 decoder both develop the same dictionary.

The algorithm starts with a dictionary that contains all the possible characters (symbols) that can be encountered in the data. In the case of letters, initially it contains all the possible ASCII characters (0-255). "*If wk exists in the dictionary then w = wk*" → it means the concatenated string is already present in the dictionary; in this case I don't output anything and I don't add anything to the dictionary. When the algorithm finds a new string never encountered ("ab"), it encodes it with the first available code (>255), and it gives as output *w*. **LZW does not immediately output the new encoded information but just *w* because the decoder will use this information to rebuild dynamically the dictionary, useful to decode the input sequence**. The first time LZW encounters a new string, it does not immediately encode it but just from the second time that the string appears in the sequence. Typically, LZW becomes efficient after the first 100 bits, with less than 100 bits Huffman may still win. Note that the number of bytes used to represent the encoded information "256" is 2, using 2 bytes to represent 3 characters it is not efficient, it becomes efficient as long as the algorithm continues to identify new concatenations of strings. When the dictionary is full, it will replace the first identified concatenation and so on every time I will find new strings.

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;
 }
```

w	k	entry/ 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			

The decoding algorithm allows to return to the original read sequences of characters. The decoder rebuilds the dictionary, as seen previously, and it uses it to decode.

Encoding (a particular case for the decoder):

Consider the string: ababbabcabbabbax

```
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		

Decoding failure:

Consider the string: ab 256 257 bc 258 262 x

```
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;
 }
```

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	??	??	??

The problem is the following: “262” encoding is not present in the dictionary! This example illustrates that whenever the sequence of symbols to be coded is *Character* + *String* + *the same initial character of the string* the decoder fails to decode it. Fortunately, this is the only case in which the above simple LZW decompression algorithm will fail. Also, when this occurs, the variable w = *Character* + *String*.

Solution:

Character + String + character concatenation is the reason of the failure of the decoder (a + bb + a + ...)

```
read a character k;
output k;
w = k;
while ( read a symbol k )
 { entry = dictionary entry for k;
   /*Exception Handling*/
   if (entry == null)
     entry = w + w[0];
   output entry;
   add w + entry[0] to dictionary
   w = entry;
 }
```

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			

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 w + w[0]; that is, *Character* + *String* + *Character*.

Up to now we have seen compression algorithms that can be used with any kind of media. Now, until the end of the course, we will see compression algorithms specific for certain types of media. We will see formats for different media that are, in reality, compression algorithms for information.

4 Images

An image is an area with a defined colors distribution. A digital image is a **bidimensional matrix**, and each point has a chromatic information. The digitalization of an image is a two-step process:

1. *Spatial sampling*: defines the resolution and the number of pixels used to represent an image. As the number of pixels increases, even the quality of the image increases.
2. *Chromatic quantization*: number of colors we can use to represent the image. It is crucial to understand that there are many and many colors, with 24 bits we can represent 60000000 colors, but we may not need all of them.

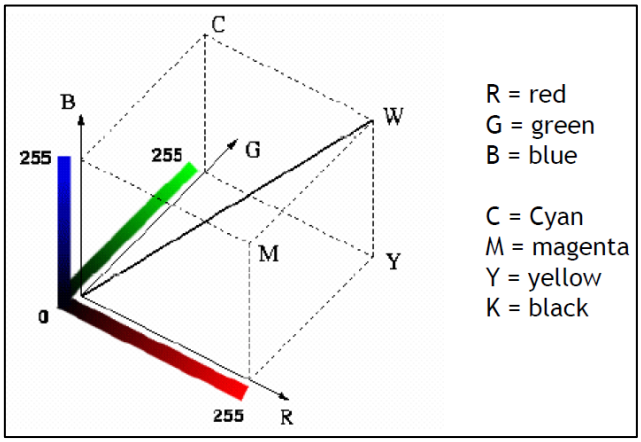
HUMAN EYE VISIBILITY

The visible spectrum is the portion of the electromagnetic spectrum visible to the human eye. Wavelength is between 380 to about 750 nanometres. The retina of the human eye has rods and cones. **Rodes** are responsible for night vision. **Cones** are of three types, responding respectively to red, blue and green colors. Our eyes are more sensible to the green rather than to the red and blue.

COLOR REPRESENTATION

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

- Additive color: **RGB** (monitor, transmitted light), where the presence of a combination describes the color.
- Subtractive: **CYMK** (printers and other devices for printing), where the absence of a component defines the color. The absence of certain colors or wavelengths of light defines the resulting color.



In both models, combining the three primary colors we can obtain other colors. Another way to represent colors is through psychophysical characteristics and this brings to a further level of distinction:

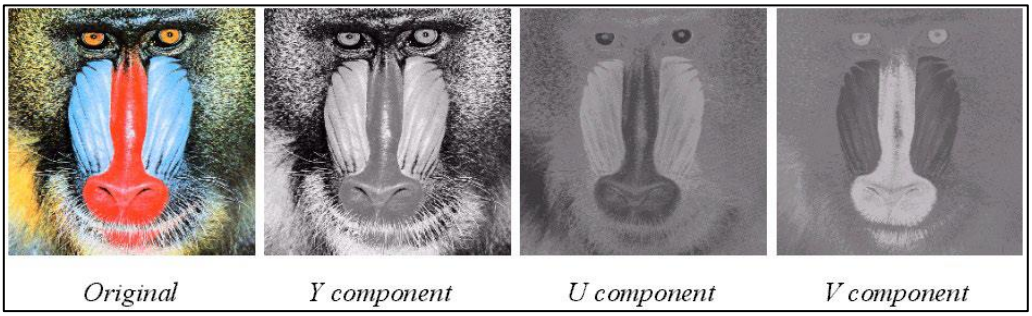
- **HSV**: it is a way to create a color and not to represent images (e.g. think about all programs that allow you to define a new color).

- **H**ue: the color itself. is described starting from the red, going through the yellow, then blue, and back to the red, in a circle like shape
- **S**aturation: how much color is present in the color itself.
- **I**ntensity (or luminosity): how much light the color contains → it is possible to manipulate this value from black to white passing through every combination between those.

Both saturation and intensity are based on the perception of human beings of shades and gradients of color. They are both described on a grey scale, and they are percentual values.

- **YUV**, **YIQ**, **YCbCr**, which separate information about luminosity (luminance - Y) from color information (chrominance - UV) → ex: TV –PAL, NTSC, digital TV.

Why do we need a format to separate the information perceived by just rodes (Y) and cones (U, V)? At the beginning the TV was in black&white, then when colored TV was invented, governments did not want to force people to buy new TVs for colored programs and so a new way to represent both types of images was needed.



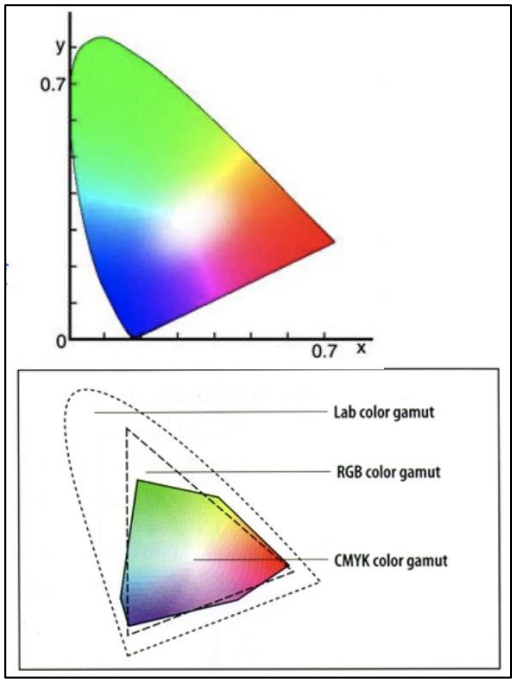
The bottom line of these models is that when we take out luminosity we gradually understand less about the image in question, meaning **we cannot only base ourselves on the colors to understand what we can lose and perceive in an image**.

CIE MODEL

Until now we described models only basing ourselves on the devices used for the visualization, monitors, printers, TV and so on. But **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**, which is a tristimulus representation system, and describe not a color but directly what our cones perceive. We describe colors based on three coordinates, in a system like so:

$$\begin{cases} x = \frac{X}{X+Y+Z} \\ y = \frac{Y}{X+Y+Z} \\ z = \frac{Z}{X+Y+Z} \end{cases}$$

Which implies that the sum of the three components is 1.



This shape is called the *horseshoe curve*, and represent pure colors, like it can be seen in picture.

The line of purples connects the extreme points of the shapes and represents colors which do not correspond to a wavelenght, and hence cannot be perceived.

At the center of the diagram there are some reference colors.

Here some images properties, useful to understand the quality of an image:

- **Color depth**
  - Monochromatic images and grayscale images (1 bit, 8 bit).
  - Indexed colors: usually 8 bit/256 colors chosen between ??? (palette or CLUT, Color Look-Up Table). If the color table is not attached in the file, the OS manages those values with its standard representation.
  - True colors: 24 bit/16.7M colors, 48 bit (High Definition), 24 + 8 bit (alpha channel).
- **Resolution**
  - Limited to the more common sizes of the screens for Web applications.
  - Images can be encoded with multiple resolutions.
  - Miniature for fast preview.

4.1 Bitmap (BMP)

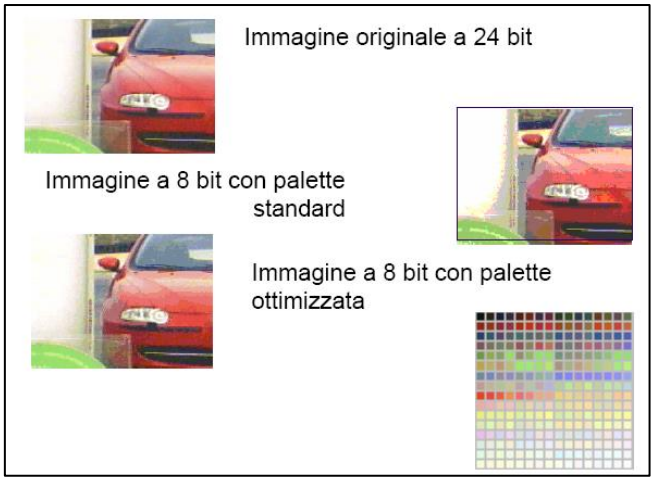
Bitmap is an MS Windows native format for images. It can represent images:

- With a palette (*colormap*) at 1, 4 or 8 bits per pixel → compressed using RLE algorithm.
- Or with natural colors using 24 bit per pixel (row format, 24 bits repeated for each bit) → not compressed.

BMP is particularly suited for artificial images or images with several sequences of equal pixels while it is not suitable for photographic images.

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: so, with 256 colors and 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 won't need to share the palette, since it is native, but different OSs will also have different palettes. Stupid as it may seem to the reader, a good example of palette is the original color system from Windows '95 Paint.



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, since the memory occupation depends on the color depth of an image, its size and the color table needed for the decompression: this quantity can range from 18.75 Kbytes to 2.25 Mbytes. Compression helps out in strongly reducing these requirements, bringing it down, in JPEG, to roughly 300 Kbytes. We are focusing on this aspect since the transfer time for not compressed images is usually not acceptable even with fast networks connections, going up to 30 seconds needed. But the intervals of time needed are even below 4 seconds when we talk about an image that just need to be visualized on the screen (the good news is the quality requirements are also lower in these scenarios). This brought the study of multimedia to create image formats, independent from the platform and the producer, and which do not require a license (e.g. GIF, PNG, JPEG).

4.2 Graphics Interchange Format (GIF)

GIF has been the first standard for image transmission over networks Developed by Unisys and Compuserve. It is characterized by 256 indexed colors and it uses the LZW compression algorithm: (patent Unisys, expired in 2004). Other properties: interlacing, transparency, animations.

GIF is suitable for simple images with few different colors (artificial images). It is a lossless compression with a fast decoding. Animated images are usually engaging and make messages more visible (advertisement). Also, it supports transparency, which allows an easy integration with the background.

4.3 Portable Network Graphics (PNG)

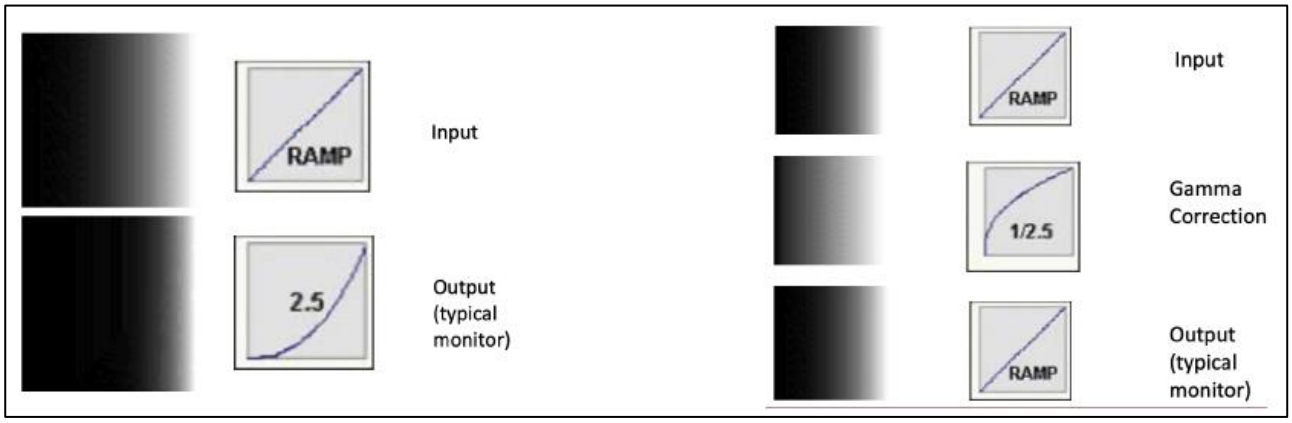
It is an extensible format, with **lossless** compression (alternative of JPEG which is a lossy compression), portable, for colorful images. No patents and licenses, born to substitute GIF. Color depth is variable from 1 to 16 bit. PNG supports indexed colors, grayscale images, and true color images. It can also manage transparency effects using an optional alpha channel and it implements an efficient interlacing method for images transmitted through the network (W3C recommendation).

Half data – without interlacing

Whole image - with interlacing

To the right we can see that we have not the best resolution possible, but we can define an image instead of having half of it shown and then wait for the entire upload.

PNG also uses a mechanism of **Gamma correction**, since, 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.



Example of Gamma correction:

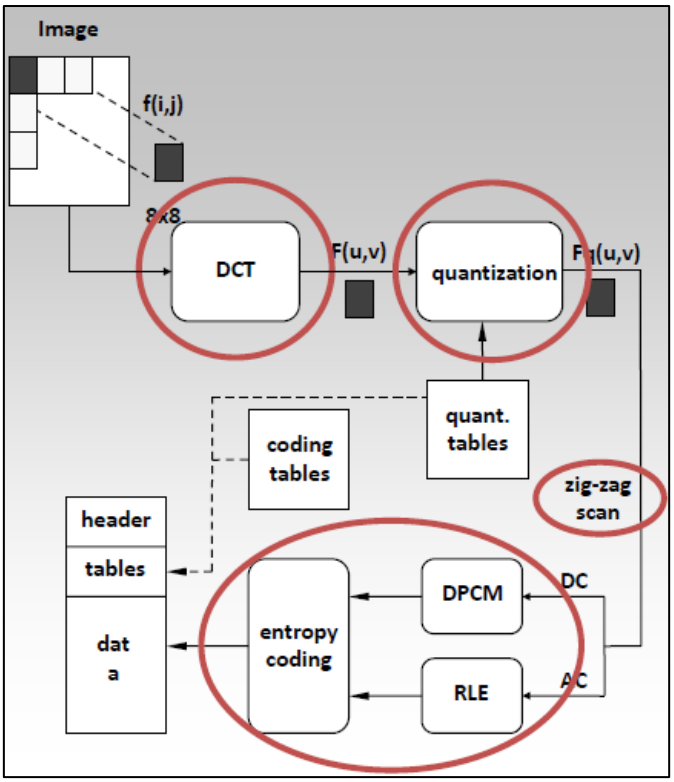
PNG has the same uses as the GIF format without requiring a license. Equal or better quality than the JPEG format, but with a lower compression. It allows a higher control of the different effects of the image through the alpha channel. It is extensible to insert additional information or proprietary information in a standard way (Promoted by W3C).

4.4 Joint Photographic Experts Group (JPEG)

It is the standard for photographic images compression. JPEG overcomes the limitation of the entropic compression, using the **redundancy of visual perception**: smallest details of an image can be suppressed without losing useful information.

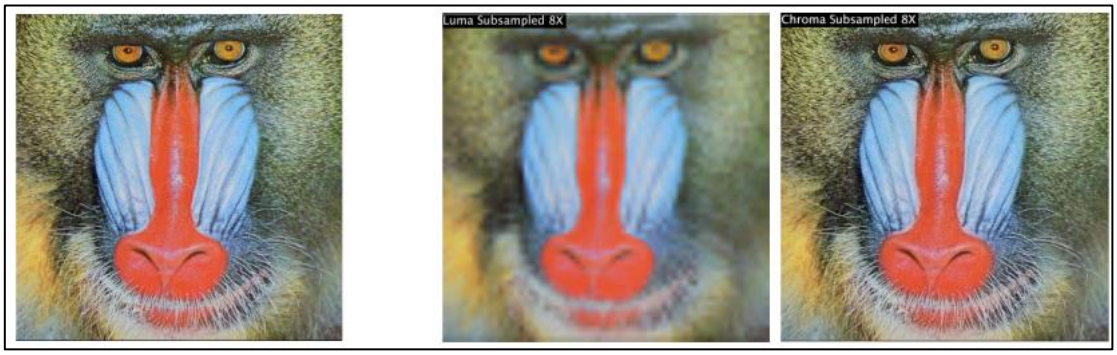
The JPEG encoding is done in six phases:

1. **Image preparation**: we switch from RGB to YUV color model (or similar). We move from a color domain to a frequencies of colors domain.
2. **DCT**: for each 8x8 block in the image, a DCT (discrete cosine transform) is applied. The DCT moves an image from the domain of color to the one of frequencies. This generates a new 8x8 matrix, where the element (0,0) is the dominant color, while the other values rapidly tend to 0. This transformation allows to understand when the information is not necessary to represent.
3. **Quantization**: each coefficient is divided by a weight defined in the quantization table.
4. **Substitution**: the value (0,0) is substituted with the difference with the same value of the adjacent matrix, in order to obtain low values. In fact, the difference of two values uses less bits instead of both values.
5. **Matrix linearization**: the obtained matrix is covered diagonally, to get the adjacent values equal to zeros.
6. **Compression**: RLE and Huffman are used to encode the resulting list of values, creating a subsampling quantization.



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

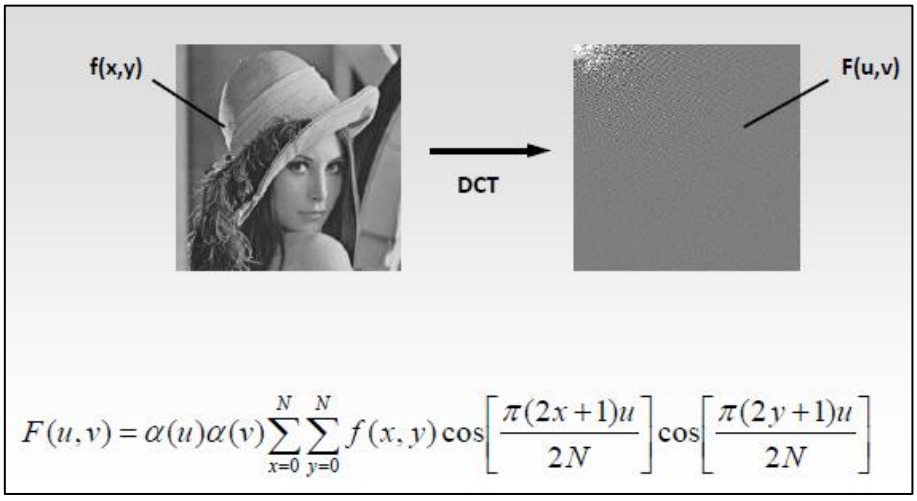
If we lose information during the preparation process, why do we switch color representation then? The human being perception is more precise for luminosity, rather than for colors: **the representation of luminance information must be more accurate than the chrominance one**, and the two components have different resolutions. So, from the YUV color coding we consider the Y (luminance), then U and V (chrominance) get encoded as differences to the reference colors. **An important detail is that we consider only U or V component, obtaining half the data with respect to the original component**. To get a better understanding, consider now the three images below, where the first is the original one, the second the one where we downgrade by 8 times the luminance, and the third one where we do the same with the chrominance. Note that the luminance decreases much more than the chrominance so luminance should be more accurate.



Now we discuss about **DCT** process. Each image is divided into 8x8 blocks. The DCT is applied to each block image *f(i,j)*, with output being the DCT coefficients *F(u,v)* for each block (as we can see from the image here below). **Each coefficient of the DCT defines the weight of the relative *spatial frequency*** (which is the rate of change of intensity or color in an image with respect to spatial position) **inside the image**. The "weight of the relative frequency" refers to how much each frequency contributes to the overall appearance of the image within the block. The element (0,0) contains the dominant color and all the other cells contain all the variations for that dominant color.

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 DCT computation very fast. **Using blocks at all, however, has the effect of isolating each block from its neighbouring context**. This is why JPEG images look choppy ("blocky") when the user specifies a high compression ratio - we can see these blocks. And in fact, removing such "blocking artifacts" is an important concern of researchers.

JPEG's approach to the use of DCT is basically to reduce high-frequency contents and then efficiently code the result into a bitstring. The term *spatial redundancy* indicates that much of the information in an image is repeated: if a pixel is red, then its neighbour is likely red also. So, **the DCT coefficients for the highest weights are the most important**. Therefore, as weight gets lower, it becomes less important to represent the DCT coefficient accurately. It may even be safely set to zero without losing so much perceivable image information.



Since the most important value to be saved is the element (0,0) of each 8x8 block and typically is the "largest" value of the block, it is possible to represent the difference between the (0,0) elements of two blocks using so less bit to represent this information.

The DCT is invertible, theoretically but in practice not so much. The arithmetic of calculator does not allow to fully reverse the DCT process so even this step loses a good amount of data. Even if the lost data are not so a large number this can represent a problem in situations where we just open the jpeg file and we save it, at every save some data are lost and, if we do this operation several times, the quality of the file decreases every time without doing anything!

After the DCT we find the **Quantization**:

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

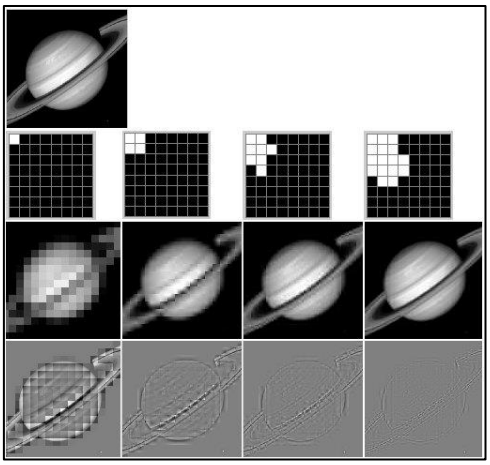
Together, these three tables work together to compress an image using the quantization process.

- The **DCT coefficients** capture the spatial frequency information of the image block. Note that not many images contain a lot of variations, and this is the reason behind all the 0 values and, since these values do not represent any useful variation, we can lose them.
- The **Quantization table** determines how much information is discarded from each coefficient; it determines the weight by which the DCT coefficients are divided: if 1 we do not lose anything, higher are these values more information is discarded. In a way, this table determines the aggressiveness of the compression algorithm.
- The **Quantized coefficients** are the compressed representation of the image block. After applying the quantization table some DCT coefficients become 0, more 0 we have more information we are losing. It is possible to undo from the quantization process, multiplying the quantized coefficients table for the quantization table.

NOTE: The result of JPEG is always more than 50% because we always lose half of the information at the beginning, with the preparation phase.

The result will be not the same of the original one. In fact, even the quantization step loses data. The inverse of the quantization is used during decompression to obtain the original DCT coefficients but all the 0s present in the quantized coefficients table after dividing them for the quantization table remain 0s because we are not able to obtain the original values (look at last 3 elements of the first row of DCT coefficients table, after the quantization process this information is lost).

Going even further in the JPG compression we see the **Matrix linearization** and **Compression**. The main idea here is that, after the quantization, we have lots of zeros in the matrix, that can be summed up with RLE encoding; a special version is applied, called “0 encoding”. The 8x8 matrix with the DCT components is linearized with a zig-zag scan. This is because 0s are not sequentially present in the matrix but there are some on the top, some in the middle and other in the last lines so we need to group them following a zig-zag scan. At the end, after RLE Huffman coding is applied.



ORIGINAL IMAGE

Number of quantized coefficients > 0, so the ones really used

Difference from the original picture

The JPEG compression has four different coding methods:

- *Sequential*: each image is encoded with one single scan from top to bottom, left to right.
- *Lossless*: it is very similar to PNG and hence not considered often since it is not so efficient. Each value is represented as the difference to the expected value based on adjacent points.
- *Progressive*: allows to show the image with low quality at the beginning and progressively with increasing quality. This is implemented through spectral selection or consecutive approximations.
- *Hierarchical*: the image is under-sampled and JPEG coded, and after the image resulting from the difference between original and rebuild is coded.

JPEG is suitable for photographic images full of colors and shades. Precise representation of minor details is not essential. Loading and visualization can be progressive. Possibility to regulate the quality of the image for visualization or printing. JPEG is the most used compression standard.

Enlarged “strelizia100.jpeg”

We can note that the orange color goes outside the flower, this is due to the choice of quantization table, we have too much 0s in the matrix. In particular what happens is this: the colors in the picture change with high contrast so the pixels in an 8x8 block change gradually

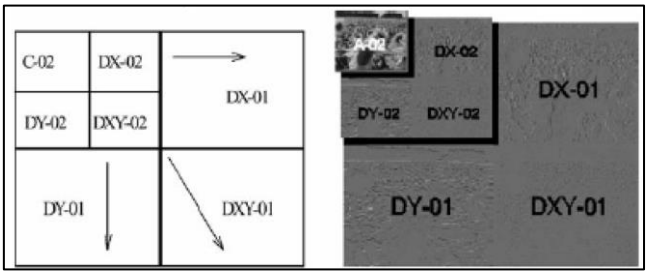
the color from a tonality to another. This is due to the fact that the dominant color is different in some portion of the block. In this case these small variations are fundamental and if we lose those values below the diagonal, we are not able to reconstruct the contrast of the original image appropriately, having so orange also outside the flower. Furthermore, when we divide that frequencies for the quantization table and we obtain 0, we are not able to go back to retrieve the original value. Note that the quality of the image is 100% and the frame above is just an enlarged portion of the original image that you can find in the didactic material → the point is to show that even with 100% of quality JPEG can fail in some cases.

4.5 JPEG 2000

It is the last compression standard for image distribution over Web and smartphones. It is not intended to substitute JPEG format but to begin a gradual transition (it may require a specific plug-in). The idea was to solve the limitations of JPEG's blocking images where the problem, as seen before, is that 8x8 blocks are independent and they do not know anything of each other, creating so the effect of the last "strelizia100.jpeg" example. JPEG 2000 is **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. It uses a **Discrete Wavelet Transform** (DWT) that provides an encoding that supports multiresolution without data redundancy, both with lossless and lossy encoding. The bitrate, as said, is way lower than in JPEG, and it supports more compression modalities and color spaces. It allows up to 256 information channels, achieving satellite images, differently from the 3 channels of RGB. To help transmission in disturbed environments (like the wireless ones) it implements a **Region of Interest** (ROI), which is a place decided a-priori that will use the most bits in the image. It is an open standard, and supports images even bigger than 4 GB, working well with both natural and artificial images. The compression systems used are the wavelet, and the DCT, allowing a full compatibility with the original JPEG.

The Discrete Wavelet Transform is different from DCT because the wavelet is applied to the whole image and not on each 8x8 blocks. When I lose data, I have not the problem to see different blocks. The result of DWT is an image divided into 4 regions (*tiles*), such that each tile dimension is half of the original image. The same procedure is repeated several times, having so 8, 16, 32, … regions, depending on how much information we want to lose.



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 idea is that the lowest frequencies are closer to the first quadrant, while going further we find the high ones.

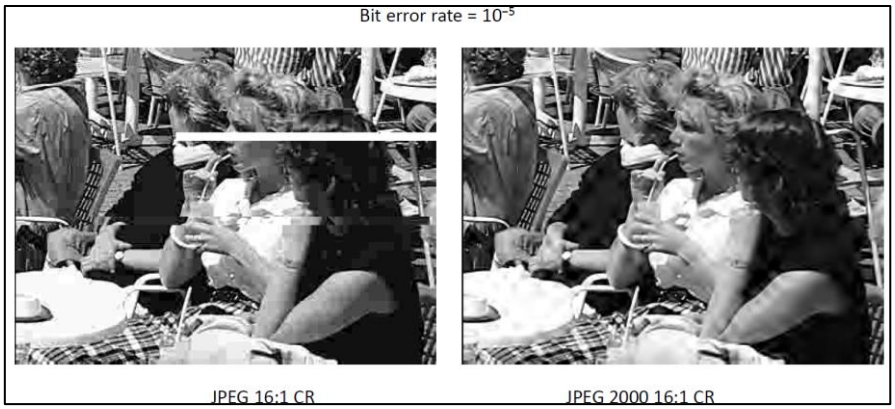
At this step we have lose some data but very few, even in the JPEG 2000 there is a quantization step. It is possible to draw a diagram of the human eye sensibility over contrast variation. The quantization coefficients are defined based on the sensibility to the associated sub-band. The final result of these processes is a better decoding phase, where even few details allow to show images less blocky with respect to the original JPEG.

Another feature that JPEG 2000 provide is the decoding and **random access**. If I want to encode and decompress just a part of the image this process allows to do that. Think about satellites images, decompress them at all will require a lot of time and that's where JPEG 2000 wins. This is related even to the fact that this format is optimized for images transmissions.

Another interesting feature, that improve the decoding process, is the **Region of Interest** (ROI). It 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 following example we have the first image without the use of ROI (not high quality) and then the one that uses ROI (again not high quality but the most important region of the image is clearly visible).

Another difference with JPEG is the following: consider the example below, in which we examine the difference of errors tolerance in the two formats, to the left JPEG, to the right JPEG 2000, for the same image, with a BER of 10^{-5}.



Looking to the left we are able to see a white stripe without color in it. That's because of the encoding of JPEG. In the transmission we lost the DC component related to the block that starts at the white strip, meaning we are not able to represent the color in the row. In a similar way, below, we lost information about another DC, but on a lower degree. JPEG 2000 manages to patch these problems, and we can see to the right the image, a bit shady but safe from clear errors.

The main limitation for JPEG 2000 is the high time required to compress and decompress, it is true that the quality is higher than JPEG, but the time required to create a JPEG 2000 image is not so short. In fact, for the websites, where load time is crucial, this format is not recommended.

4.6 Vector Graphics

Pictorial images are represented as matrixes of points (pixel); 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. Pictorial images need to be acquired; artificial images are the result of elaboration from the calculator. The main difference is that the images are only described by a mathematical formulas. For instance, these images have not problem of zooming in, the formula is just multiplied to see deeper.

Vector graphics allow high precision for detailed draws. Quality is independent of pixel. This format provides a compact dimension and details enlargement without quality loss (scalability). Images are also easy to manipulate.

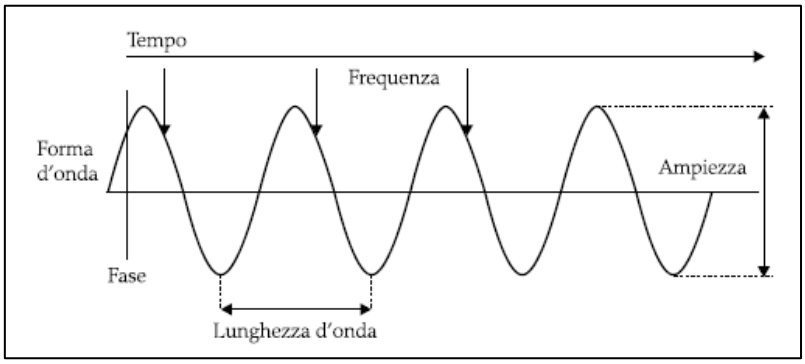
Vector graphics are suitable for:

- graphics and paper advertisement because easily editable and scalable;
- urban and construction design and for industrial design (3D graphics);
- typography (Typeface description);
- animation (videogames).

5 Audio

As we have done for images, understanding how our eyes work, we need to discuss about how our ears work. There are sounds that we are able to perceive and others that we are not able to hear. But what is a sound? The sound is a longitudinal pressure wave (generated by a movement) that propagates through a transmission medium (the air). The range of human hearing is between 16 Hz and 22 kHz. The sound is formed by:

- **Amplitude** (dB) → in terms of perception is the volume, the intensity of the sound. Higher is the amplitude higher is the volume of the sound.
- **Frequency** (Hz) → perceived as "higher" or "lower" sound. Note: different from saying "higher/lower volume" which deals with amplitude.
- **Waveform** → timbre, it allows us to recognize different types of sound production.



Intensity levels

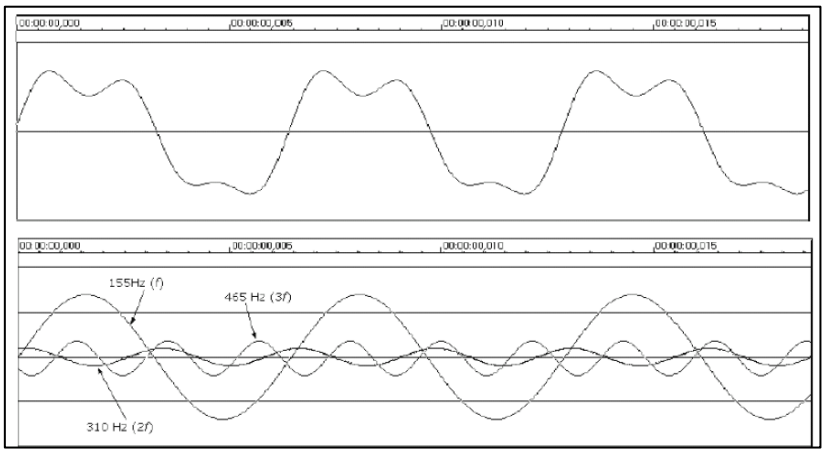
Approximate sound levels and intensities within human hearing range

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}	

Note that 0 is just used as a threshold and it does not represent the state of silence. When we hear a sound, it is just because the noise is over 0.

The **Fourier analysis** gives us an important result: "*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.*"

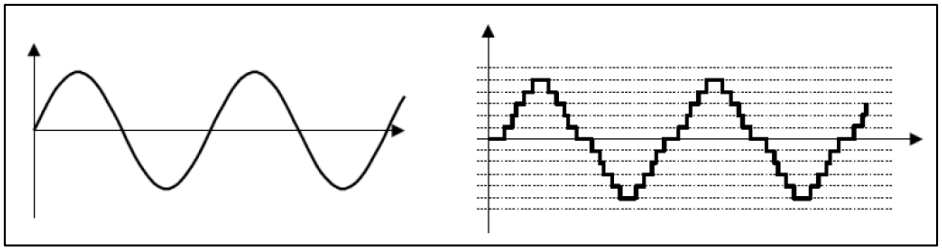
The point of this analysis is that we can decompose a complex signal into the sum of easier signals to analyse. Consequently, we have more signals to analyse but they are easier than the original one. Here below there is an example of this:



The first step to elaborate a signal is to digitalize it (A/D transformation)

- ***Sampling***: the reduction of a continuous-time signal to a discrete time signal. A common example is the conversion of a sound wave to a sequence of "samples", where a “sample” is a value of the signal at a point in time and/or space.
- ***Quantization***: discrete representation of the signal level (measured in bits of precision).
- ex. Audio CD is digitalized at 44.1 kHz, 16 bits.

The following example shows the digitalization of a signal:



How much time I need to sample the signal? Another result, **Nyquist Theorem**: “*if a periodic signal contains no frequencies higher than N hertz, it can be completely reconstructed if 2N samples per second are used. So, a sampling-rate of 2N samples per second is sufficient*”.

If the samples are not enough the wavelength is not reconstructed in the best of ways since we can hear the difference from the original audio. The right number of samples to use to represent a signal has to be a number that reduces at the minimum the difference between the original audio and the reconstructed one.

If we do not use the right number of samples we can add noise, so we need to be careful in respecting the Nyquist Theorem. 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.

$$SNR = 10\log\frac{V_{signal}^2}{V_{noise}^2} = 20\log\frac{V_{signal}}{V_{noise}}$$

In digital systems, the noise is the difference between the real signal and the quantized signal (digitalized signal).

$$SQNR = 20\log\frac{V_{signal}}{V_{quant-noise}} = 20\log\frac{2^{N-1}}{1/2} = 6.02N\,(dB)$$

2^{N-1} is the maximum value of the original audio and ½ is the maximum value that we can add with digitalization. The result of the above formula can be interpreted as follows: **every used bit for the encoding of samples adds 6.02 dB to the final SQNR** (1 bit = 6.02 dB, 2 bits = 12.04 dB, etc.).

Audio quality vs size

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

Consider that these numbers depend on people's age, sex and other characteristics. The above number are an average, but they may be not true for all people.

Digital encoding and decoding of audio signals are more problematic than the image encoding: the temporal structure of the audio cannot be modified (frequency), and the audio information varies over time ("audio stop" does not exist). Also, the required reproduction quality goes beyond the simple understandability of the message. As for the images though, the uncompressed audio strategy is not suitable even for conventional networks, since it would require for FM Radio quality 1.2 Mbit/sec. This is due to audio's high dimension (coded signal takes a lot of space and the length is not limited → live audio), since, for example in a streaming scenario, it is not possible to transfer the whole audio file before playback.

There are many different audio formats that have been developed over the years, some of which are:

- **Waveform Audio File (WAV)**: developed by Microsoft and IBM, it is the standard used for audio encoding on PCs, and it does not use compression.

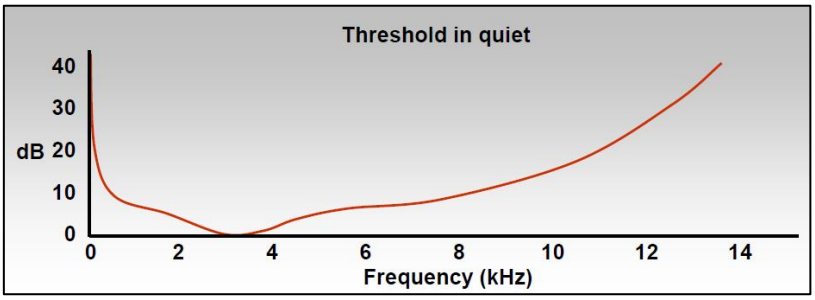
- **Audio Interchange File Format (AIFF)**: developed by Apple, it is the standard for MacOS, and, again, although it supports a compressed version, it is not compressed.
- **μ-LAW**: standard audio format for Unix, it is also the telephonic standard of the USA.
- **A-LAW**: European version of μ-LAW.
- **MPEG-1**: it encodes audio tracks in MPEG-1 videos, meaning it is a compressed format for variable quality encoding. The algorithm works on several steps based on the psychoacoustic principles and allows three different encoding levels with three different bit-rates (the standard MP3 is the MPEG-1 at level 3). It is used as the cross-platform standard, since several applications for consumers market use it widely.

Let's do some crucial consideration before talking about MPEG-1 compression. **Lossless compression for audio is not feasible**, since it provides really low performances: audio data are extremely variable, and it is rare to find repeating patterns in it. Moreover, audio information is redundant, meaning we can control the compression quality to please the human ear.

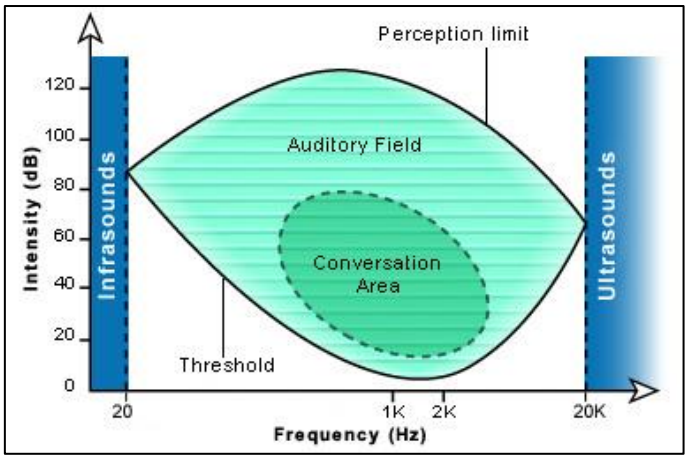
Some kind of lossy compressions that use very few data are:

- **Silence compression**: we identify silence as a consecutive set of samples under a defined threshold and compress it with the philosophy of RLE.
- **Adaptive Differential Pulse Code Modulation**: encoding the difference between consecutive samples allow to have a quantization of the difference, meaning we can lose information which is not meaningful.
- **Linear Predictive Coding**: we adapt the signal to a human speech model, and transmit the parameters of the model and the differences of the real signal to the model.

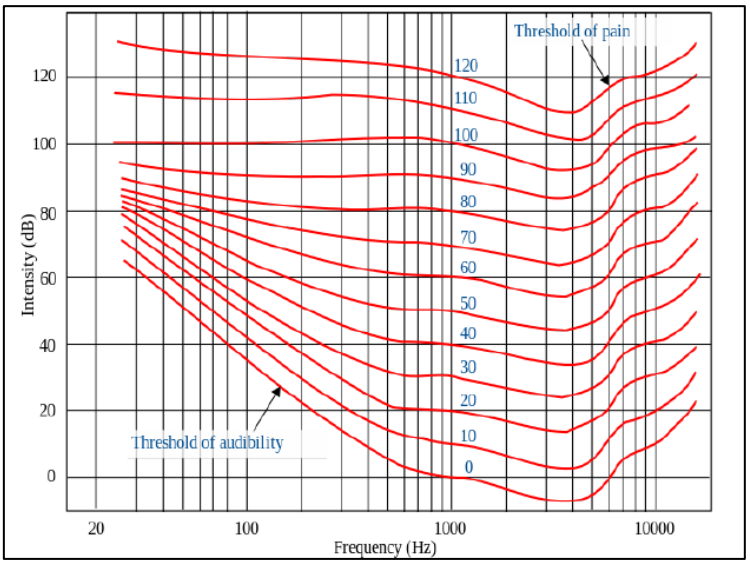
But before actually applying these techniques of compression, we need to understand what we can/cannot lose, and we can do it studying psychoacoustics elements of a sound. The human ear sensibility varies along the audio spectrum: the maximum sensibility is around 2-3 kHZ (***conversation area***) with amplitude>0 and decreases at the spectrum extremities. We have to keep in mind the sensibility is deeply influenced by personal factors. The below graph shows the level of sounds, considering the couple (amplitude, frequency), that we are able to perceive (those above the curve). So, everything below the curve can be not stored because we are not able to listen it and so that tones are useless elements for audio compression.



Humans perceive sound and voice at a hearing interval of 20Hz to 22kHz, whereas **the *recognizable dynamic interval***, which is the interval between the weakest and strongest sound, **is roughly 96 dB**.



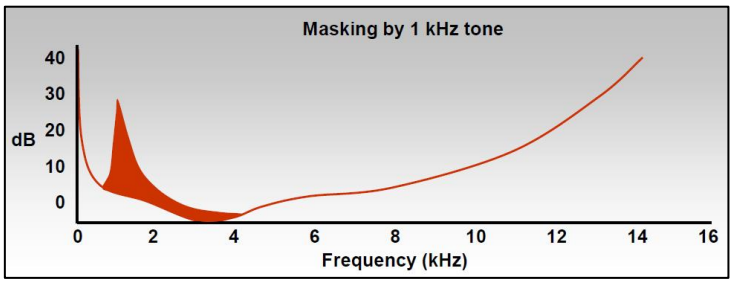
Fletcher-Munson Curves



An instrument that helps clarifying an aspect of human earing are the Fletcher-Munson curves, which are displayed above. We can see thresholds for hearing and pain, but the main focus is on the part between two red lines: in there, we perceive the sound at the same volume. These 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, and the spectrum can be divided in 25 of them. Doing this, we can consider the human hearing as a series of overlapping ***band-pass filters***.

All the above considerations are true if we listen only a sound with silence, but this is not generally true. The main issue now is what happens when sounds overlap (**tonal masking**): **a pure sound can mask another with near frequency and lower volume**, but, during the playback of a sound at 1 kHz, other simultaneous sounds in the masking interval cannot be perceived. This is different from **non-tonal masking**, where we involve a sound that has a higher intensity than the masking sound (but the masking one is still able to silence the other).

Furthermore, with the **temporal masking** a sound can mask another one for a small interval of time. The *pre-masking* hides preceding sounds between 5ms and 40ms before. The *post-masking* hides the weakest sounds after the masking sound between 50ms to 200ms.



The above graph shows what happens when a sound mask another, in particular with a masking by 1 kHz tone: we are no more able to hear the highlighted red area, even if it is above the curve of perception.

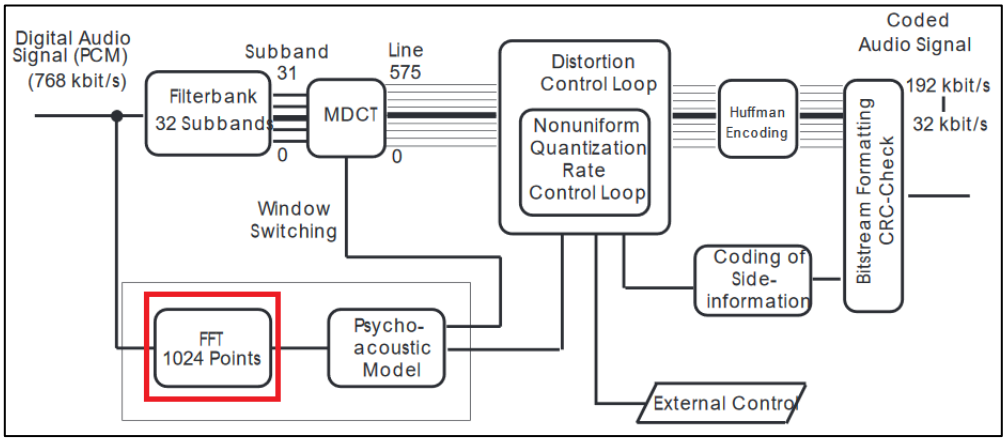
5.1 MPEG-1

MPEG-1 layer 3 (MP3) is the current standard for high-quality audio (music, in general) with compression. The most common bit-rate for MPEG standard is from 48 KB/sec to 384 KB/sec, and, even though its compression level is 6:1, it is able to obtain a result almost identical to the original signal. From 96 to 128 KB/sec, it represents the best quality for consumer applications, since it also supports monophonic, dual (two different channels), stereo and joint stereo signals. MPEG-1 uses different sampling frequencies (32, 44.1 and 48 kHz → note that we cannot perceive these frequencies, but we can perceive how they influence the waveform). MP3 is a transparent encoding, with a transparency threshold of 2.1 bits/second, from which we are not able to distinguish the original file from the compressed one.

MPEG audio compression algorithm works on four 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 (useless).
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 (1 bit is close to 6 dB of noise).
5. Creates the bitstream following a standard format of transmission.

The entire encoding process is summarized by the following schema:



Following the above schema, the more precise process of MPEG Audio Encoder is the following:

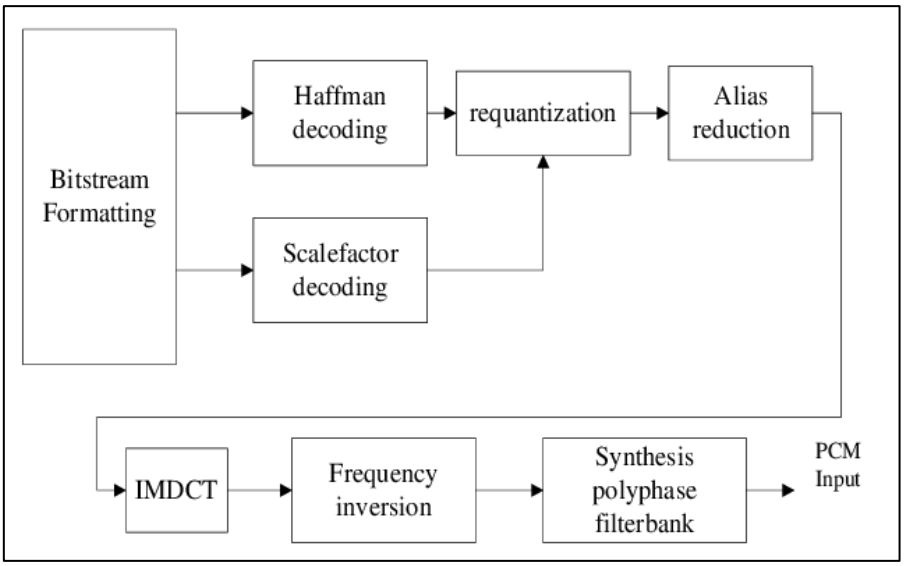
1. **Digital Audio Signal (PCM) Input**: the process begins with a digital audio signal in Pulse Code Modulation (PCM) format, typically at a high bitrate of 768 kbit/s.
2. **Filterbank (32 Sub-bands)**: the input audio signal is passed through a filterbank that splits the signal into 32 sub-bands. This step helps in separating different frequency components of the audio signal, which is crucial for further processing.
3. **FFT (1024 Points)**: a Fast Fourier Transform (FFT) with 1024 points is performed. This step transforms the time-domain signal into the frequency domain, which is necessary for analysing the frequency components of the audio signal.
4. **Psycho-acoustic Model**: it uses the output of the FFT to analyse the auditory masking effects. This model determines which parts of the audio signal can be removed or reduced without significantly affecting perceived audio quality. It uses principles of human auditory perception to identify and discard inaudible or less important information.
5. **Window Switching**: this step involves dynamically switching between different window sizes during the transformation process to handle transient signals better and to improve the compression efficiency and audio quality.
6. **MDCT (Modified Discrete Cosine Transform)**: the subbands are then transformed using the Modified Discrete Cosine Transform (MDCT). MDCT provides higher frequency resolution and is used to further process and compress the audio data by converting time-domain data into frequency-domain coefficients.
7. **Distortion Control Loop**: this loop ensures that the quantization noise introduced during compression is controlled and remains below the masking threshold identified by the psycho-acoustic model. It adjusts the quantization process to maintain audio quality.
8. **Nonuniform Quantization Rate Control Loop**: this step applies nonuniform quantization, meaning different parts of the audio signal are quantized at different rates

based on their importance and the psycho-acoustic model's guidance. It helps in efficiently compressing the audio data while maintaining perceived audio quality.

9. **Huffman Encoding**: the quantized coefficients are then encoded using Huffman coding, reducing the overall size of the data.
10. **Coding of Side-information**: alongside the main audio data, side-information is encoded. This includes information necessary for decoding, such as bit allocation, scale factors, and other parameters required for proper reconstruction of the audio signal.
11. **Bitstream Formatting and CRC-Check**: the final step involves formatting the encoded data into a bitstream that can be stored or transmitted. A Cyclic Redundancy Check (CRC) is included to detect any errors in the transmitted or stored data.
12. **Output - Coded Audio Signal**: the result is a coded audio signal with a reduced bitrate, typically between 192 kbit/s and 32 kbit/s, which significantly compresses the original audio data while maintaining a high level of perceived audio quality.

The entire process leverages psycho-acoustic principles and efficient encoding techniques to compress audio data, making it suitable for storage and transmission while preserving sound quality.

The decoder step is represented by this other schema:



MPEG is an <u>asymmetric</u> compression. When I encode, I can use more time than the time used for decoding. In asymmetric compressions the goal is to have a very fast decoding → users should not wait so much for play an audio. So, the encoding can require more time and that's why the encoder uses two different maths processes to decompose the signal in a different way (the above part of the schema and the FFT which requires most of the computational resources). The final quality of the audio, that users listen to, is determined by the Psycho-acoustic Model and that's why is so crucial to use FFT transformation.

Consider the following example: the masking level for band 7 is 12 dB, 15 dB for band 9. If the level at band 7 is 10 dB it gets ignored in the compression, since it's lower than 12, while if at band 9 we find 35 dB it ends up compressed. Remember that only the difference between the signal and the masking threshold is encoded in the end.

Another useful application of masking is the one of **watermarking**. This is the inclusion of digital information (like source, destination, copyright, information, and so on) hidden inside multimedia data like images, videos, audio text and even animations. The watermarks cannot be modified and do not have the necessity to modify the enclosing data. Watermark insertion is done in an audio by doing the opposite to the MPEG algorithm: you insert the watermark near high-level signals, such that it gets masked, and it is not distinguishable from the original one.

There are three different MPEG audio 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.

- **Layer 2** (bitrate equal to 128 Kbps): works with three frames during the filtering process (previous, current, next, 1152 samples in total).
    - Partially works with temporal masking.
    - Uses a more compact representation of additional information (header, number of bits for each band, ...).

- **Layer 3** – **MP3** (bitrate at 64 Kbps): divides the frequencies spectrum into different sub-bands with non-equal amplitudes (shorter when we are more sensible and longer when we are less sensible), more comparable to the critical sub-bands in the lower frequencies.
    - Psychoacoustic model with temporal masking (50ms before the sound and 200ms after the sound).
    - It considers stereo redundancy → only the difference between the left and right channels.
    - Variable bitrate:
        - Huffman compression with pairs of values.
        - Uses a ***bits reservoir*** (a container of useless bits) with bits from frames that do not need them and can be allocated to the frames that do.

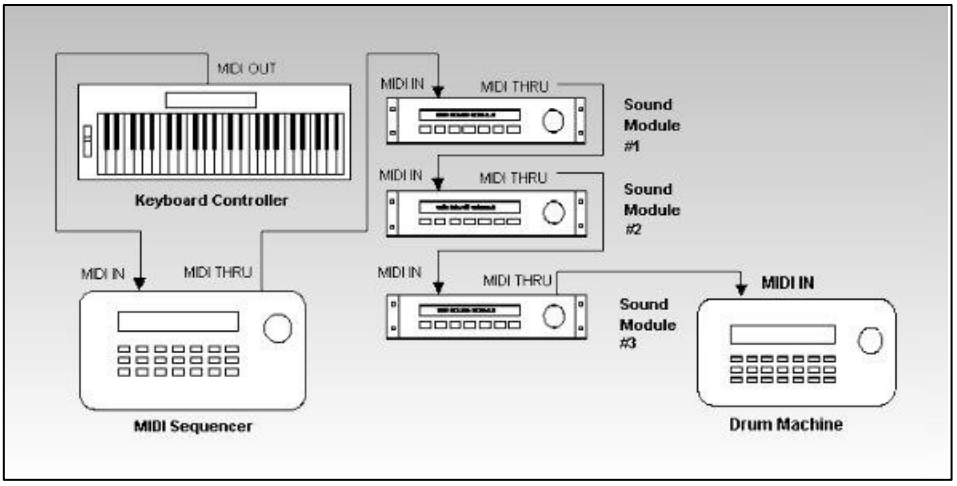
Transparent encoding: we are not able to recognize the audio compressed file from the original one. This if we have a 2.1 bits per sample (multiply for frequencies of samples) we have a transparent encoding, below this number we don't have a transparent encoding. MP3 allows to have a bit rate of 64kBs and due to this MP3 is not a transparent encoding.

The MPEG layer 3 (MP3) is now the most used standard and following it also MPEG2, in 1994, and MPEG4, in 1999, were developed. The first is the standard for DVDs and was created aiming at transparent sound reproduction for theatres. The latter consider audio as a composition of different objects, where the user can decide to listen a composition in different environments, or even emphasize some components more than others.

5.2 MIDI

MIDI (Musical Instruments Digital Interface) protocol dates to 1983 and provides a standard and efficient way to describe musical events, in the same ways a score is used for real-life instruments. It is really different from what we have seen with MPEG: it does not exploit wavelengths but scores (no noise). It enables computers, synthesizers, keyboards and other musical devices to communicate each other via a scripting language that codes ”events” (notes for instance). Sound generation is local to the synthesizers and messages describe the type of instruments, notes to play, volume, speed, effects and so on. MIDI on itself is a pretty complicated system, as the below example suggests, but most of modern soundcards come with all the necessary hardware. While in MP3 we speak about quality referring to the psycho acoustic model, the MIDI quality is defined by its synthesizer component.

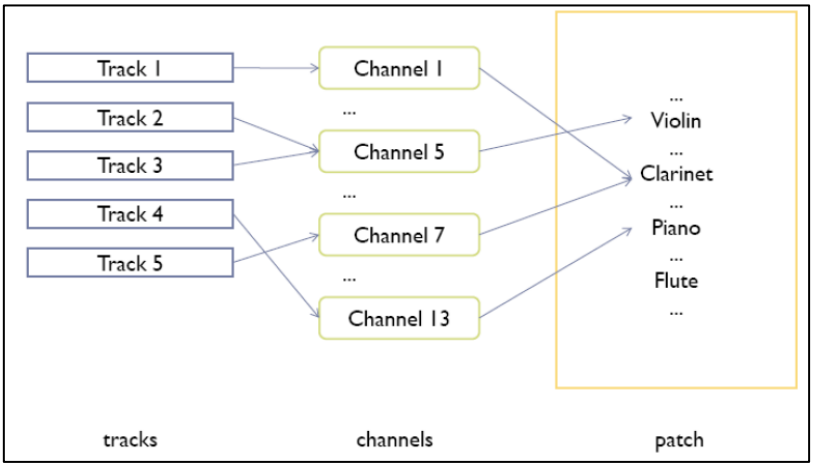
Example of a MIDI system



In the above system we can note an interesting component: the **MIDI Sequencer**, which is 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 (like speeding up the tempo of a song, not possible in MP3 without altering the frequency itself → easy to do in MIDI: basically, we have just to send a message!) and creates the music sending data to devices like sound cards. MIDI has no influence on the quality of the sound during this process, which is entirely reliable on the **synthesis device**.

MIDI is organized through:

- **Channels**: allow to send and receive music data (MIDI events). They are a method to differentiate timbres and send independent information, this implies different channels are used for different instruments. MIDI channels start from 1 to 16: I can play only 16 instruments in the same moment (think about classical music, this is a limitation).
- **Tracks**: a structured autonomous flow of MIDI messages (an instruction to the system: for instance, a note), like in a piano song, where there are a melody and an arrangement. Each track is associated with one or more channels (a track can be played by more instrument at a time).
- **Patch**: specifies the timbre produced by the generator, MIDI can contain up to 128 different patches, which contain pitches. Each channel is associated with one patch.



MIDI messages are divided in **channel** ones, which describe which note to play (*voice*) and how to play it (*mode*), and **systems messages**, which define set-up and synchronization information. Channel messages refer to a particular channel while system messages refer to the whole system. Each message requires 10 bits (1 byte + 2 bits): 8 bits used for the message itself and the other 2 are the payload. Furthermore, there is 1 bit at the beginning and 1 bit at the end of each message to separate different messages. At the end each message requires 12 bits. The 8 bits of the message are divided in two nibbles of 4 bits, where the first represent the message type (channel or system), while the second the number of the channel to which the message is directed: we use 4 bits and that's why MIDI use only 16 channels.

Channel messages contain the number of the channel through which the information is sent and they both include:

- ***Voice messages*** define what an instrument play:
  - 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
    - Poly/Mono
    - General MIDI Mode

System messages do not use a channel because they are meant for commands that are not channel-specific. Each device responds only to the messages it is enabled to answer. These messages include:

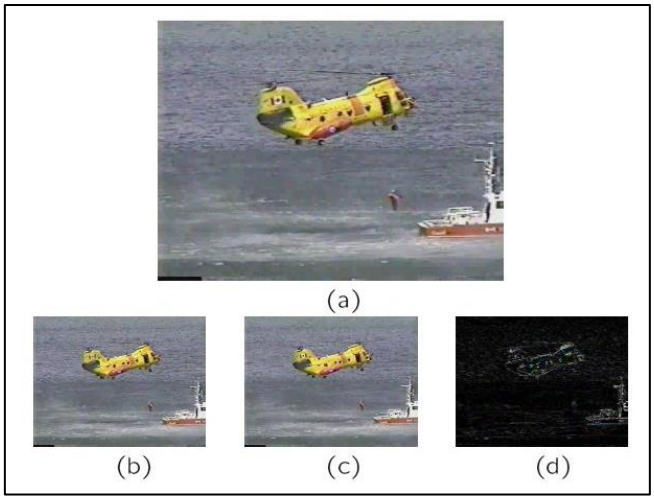
- ***System common messages***
    - Carry out general functions that involve the entire system (ex., song synchronization when played by different devices)
    - Set up a common clock
    - Positioning inside a song (Song Position Pointer)
    - Track selection (Song Select)
- ***System real-time messages***
    - 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.

MIDI is an efficient standard to encode musical sounds inside web documents, since its compact and does not depend on the waveform of the sound. It is particularly well-suited for background music. But it can only represent traditional western music, and sounds like voice or other acoustic phenomena are not encodable. Also the quality, as said, depends explicitely on the quality of the audio card of the device, which needs a specific architecture to use MIDI in the first place. Also channel and message coding is not completely standard, since it may vary, for example, between synthesizers.

6 Video

An **analogic video** is encoded as a continuous signal that varies over time, meaning it can be digitalized but not further elaborated, due to the bi-dimensionality of the image. A **digital video** is a sequence of digital images, where we see a continuous color information flow: in this context 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, ***fusion of flicker***, 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 allows us to perceive a video**.

A first example of digital video can be seen in an **interlaced video**, where each frame is divided in two types of rows, odd lines (of the previous image) and even lines (of the new image), which are shown alternatively. According to the number of rows we can achieve from 25 to 50 frames per second using less data. Here below an example:



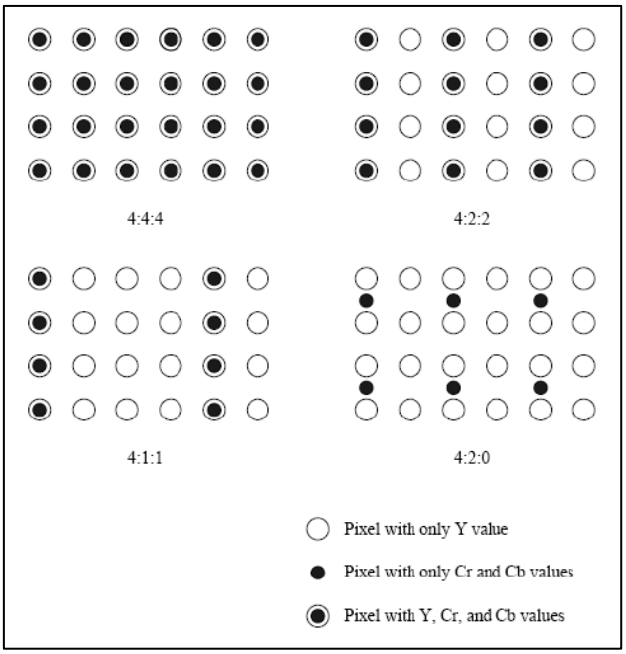
Other definitions of digital video can be drawn, depending on the types of signals they require:

- **Video with separated components**: each primary signal, like RGB and YUV, is transmitted separately, allowing a better color reproduction due to the absence of interference between signals. This, although, requires high bandwidth due to the precise synchronization.
- **Composite video**: luminance and chrominance signals get mixed in a single carrier wave, and, differently from before, we see interference.
- **S-Video**: chrominance signals are mixed in a single carrier wave, while luminance ones are sent separately.

Analog video usually uses a composite signal (always for transmission). Digital video uses a signal with separated components.

The transmitted videos trough these signals have properties, which are related to different characteristics. First, the **color depth**, which is usually a true-image, due to the recording. Then, the **resolution**, which depends on the standard of the video, while the chrominance gets undersampled → remember that higher resolutions require more space, like the example of 4K videos which need up to 5 GB per second. Finally, the **frame frequency**, which depends on the region in which videos are transmitted: PAL uses 25 frames per second, while NTSC roughly 30 frames per second. The minimum to avoid the perception of snap movements is 15 frames per second. Usually, NTSC is referred to north America and Japan, while PAL to Europe and most of Asia.

In this context we find the ***sampling***, which is a ratio that describes how much information we have about a video: with a 4:4:4 we have all the information at any time for each pixel, while as for the 4:2:2 we find, every four pixels, 2 pixels of information about chrominance, 4 about luminance, and 2 about Cr and Cb values. Other combinations are 4:1:1 and 4:2:0 (we store all the information about luminance and every 2x2 block we compute the average for blue and red chrominance for that 4 pixels). This is useful, for instance, to manage colored and black&white televisions, only changing the sampling technique.



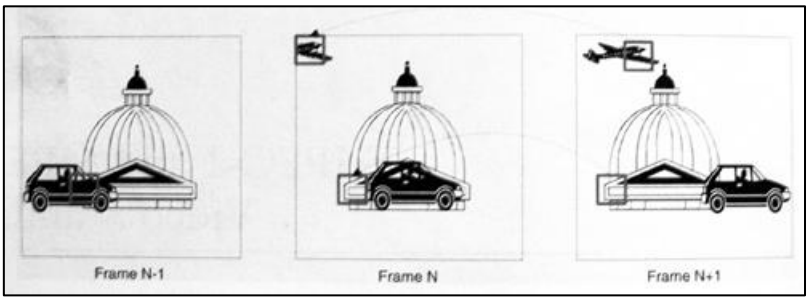
The uncompressed video requires a considerable amount of storage: High-Definition Television (HDTV) requires a bit-rate that can be higher than 1 Gb per second, meaning we must compress data: 1 hour of MPEG-1 video with a VHS requires roughly 600 MB, which leads to a lossy compression technique made by two steps:

1. **Elimination of spatial and temporal redundancy**.
2. **Intra-frames *(I-frames)* and inter-frames *(P-frames)* encoding**: where the first refers to the compression of single images, the second the compression of data due to redundancy (encoding the difference between two images).

The memory usage highly influences the transfer time of a video over a network, which has the same problems as the images transmission, combined with the fact that it is a continuous medium, it must follow a constant frame-rate and the loading time must be compatible with the reproduction time. On this latter point, a download and play solution, nowadays, is not always applicable, which is why we find ourselves in a streaming scenario. This means that, rather than a server giving all the data to a client, we will have a gradual transmission of data in a buffer, that has to be consumed in order to receive more.

6.1 Motion JPEG

It is the first attempt of digital video format, where the video signal is encoded as a sequence of frames, and each frame is encoded as a JPG image, meaning it does not take advantage of the clear correlation between one frame and the other. When encoding video frames (*intra-frame*), 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 (*inter-frame*). The following example shows that is possible to reconstruct the frame N using the information of the previous and next frames, respectively N-1 and N+1.



The differences between one frame and the next one usually depend on the movement of some pieces of the frame. So, the only thing we are going to store are the differences between images in time. This is done by storing the index of an image, compressing a lot of redundant data, and achieving a lossy compression, which is corroborated by the fact that these index frames are in JPEG format. We hence lose data from chrominance, the JPEG compression, and in the differences in the two frames. Video compression algorithms that adopt this approach are said to be based on **motion compensation**. The three main steps of these algorithms are:

1. Motion estimation (motion vector search).
2. Motion compensation-based prediction.
3. Derivation of the prediction error - the difference.

For efficiency, each image is divided into macroblocks of size NxN. In fact, motion compensation is not performed at the pixel level but instead at macroblocks level. The current image frame is referred to as the **Target frame**. A match is searched between the macroblock under consideration in the Target frame and the most similar macroblock in previous and/or future frame (referred to as **Reference frame**). In that sense, the **Target macroblock is**

predicted from the Reference macroblock. The displacement (or movement) of the reference macroblock to the target macroblock is called **motion vector**. The difference of the two corresponding macroblocks is the prediction error. Here below an example of how this works:

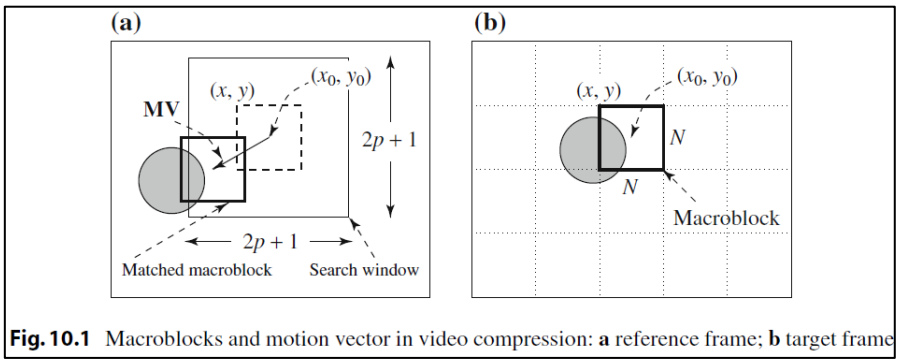


Fig. 10.1 Macroblocks and motion vector in video compression: **a** reference frame; **b** target frame

For video compression based on motion compensation, after the first frame, **only the motion vectors and difference macroblocks need be coded**, since they are sufficient for the decoder to regenerate all macroblocks in subsequent frames.

The search for motion vectors is a matching problem. Anyway, calculating the movement of objects (so, the motion vectors) requires lot of time. The difference between the two macroblocks can then be measured by their ***Mean Absolute Difference*** (MAD), defined as:

$$MAD(i, j) = \frac{1}{N^2} \sum_{k=0}^{N-1} \sum_{l=0}^{N-1} |C(x + k, y + l) - R(x + i + k, y + j + l)|$$

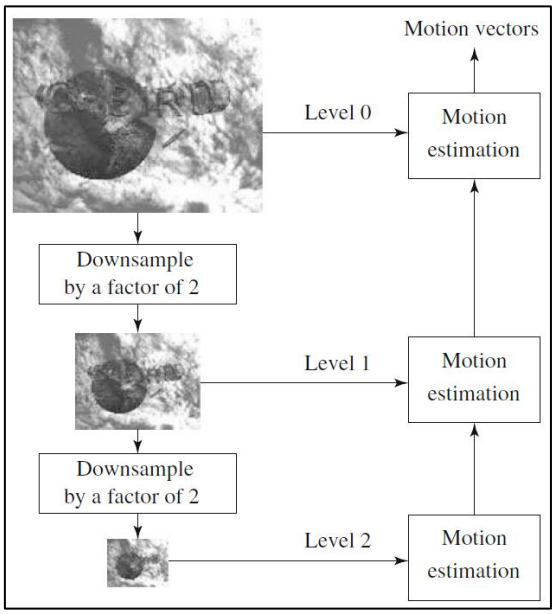
The goal of the search is to find a vector (i, j) as the motion vector MV=(u, v), such that MAD(i, j) is minimum. (N is the size of macroblocks.)

NOTE: what makes different a symmetric compression and an asymmetric compression is the size of both search area and macroblocks, so the amount of computation required to encode. Larger is the search area and smaller are the macroblocks, more computational time to encode is required. The computation depending even on the algorithm used to search a match between macroblocks → Sequential (best precision) vs Hierarchical Search (less precision).

The motion vector can be calculated by the **Sequential Search algorithm** (Full Search), that is computationally very expensive, going up to $O(p^2N^2)$, and creating hence an asymmetric encoding, since this analysis won't be needed in the decoding phase. The high complexity is given by the fact that the algorithm compares a macroblock centered at each of the positions within the window with the macroblock in the Target frame, pixel by pixel, and their respective MAD is then derived using the above formula. The vector (i, j) that offers the least MAD is designated the MV=(u, v) for the macroblock in the Target frame.

The search for motion vectors can benefit from a hierarchical (multiresolution) approach (**Hierarchical Search algorithm**) in which initial estimation of the motion vector can be

obtained from images with a significantly reduced resolution. The below figure depicts a three-level hierarchical search in which the original image is at level zero, images at levels one and two are obtained by downsampling from the previous levels by a factor of two, and the initial search is conducted at level two. The initial estimation of the motion vector is coarse because of the lack of image detail and resolution. It is then refined level by level toward level zero. This approach allows to compare the target frame with only the macroblocks near to the downsampled image.



After having calculated the motion vector (***motion estimation***) the idea is to use it to rebuild the frames that I'm encoding moving the current macroblocks using the motion vector itself. Then we compute the difference between the actual current frame and the new predicted one (again, the result of the application of motion vector over the current frame). This difference contains the information that was not predicted accurately by the motion vectors → this process is called "***motion compensation***". At the end we save only the motion vector and the differences (residuals), elements used by the decoder to decompress the video.

The first video standard, developed for videoconferences, is **H.261**, which encodes very small images (352x288), and is also called pstarch, due to the fact it has p channels (up to 30) all sending 64 Kb per second. It develops a symmetric encoding, since encoding and decoding must happen in real time with a maximum delay of 150 ms. The output is in slow motion, since it varies between 10 and 15 frames per second, while the input is in roughly 30 frames per second. Intra frames are treated as independent images and use the algorithm of JPEG (not the standard, to avoid the headers), while the inter-frames are estimated with motion estimation and compensation. P-frames are not independent. They are coded by a forward predictive coding method in which current macroblocks are predicted from similar macroblocks in the preceding I- or P-frame, and differences between the macroblocks are coded. Temporal redundancy removal is hence included in P-frame coding, whereas I-frame coding performs only spatial redundancy removal. It is important to remember that prediction from a previous

P-frame is allowed (not just from a previous I-frame). The interval between pairs of I-frames is a variable and is determined by the encoder. Usually, an ordinary digital video has a couple of I-frames per second. Motion vectors in H.261 are always measured in units of full pixels and have a limited range of ±15 pixels - that is, p = 15.

NOTE: An important note is that the H.261 encoder contains even a decoder. This is due to the JPEG compression of the frames. In fact, when I need to encode the second frame, which is a P-frame, I cannot search for macroblocks in the original frame because it does not contain all the information that I really have in my file. Remember that JPEG is a lossy compression. The search of macroblocks happens in the original frame encoded and decoded.

The standard **H.263** instead, uses PB-frames for inter-frames compression, meaning it is predicted and bidirectional: the difference is hence calculated from both the previous and following frame. Once calculated the differences between two frames only direction and movement are transmitted, which is the motion vector. **H.263 allowed the motion vector to refer to pixels outside the boundaries of an image** (*unrestricted motion vector mode*), associating the nearest pixel to the edges of the image, the one pointed by the motion vector, to the outside of the image. In fact, this is realistic since some part of the image can "go out" from the image borders. This achieves a better quality with more time consume, which were the reasons that brought to the creation of MPEG.

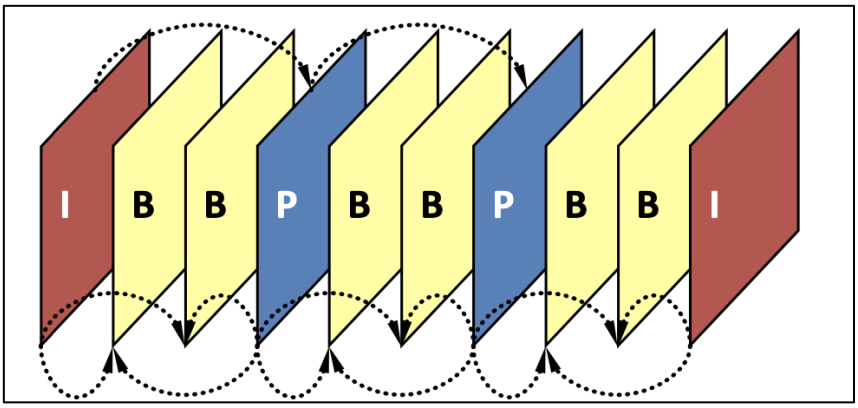
6.2 MPEG

The first MPEG dates back to 1991, and allows the compression of a sequence of images, stored on a CD, with the aim of random video access and fast research. The compression algorithm is very complex, but most importantly **strongly asymmetric**, since it assures real time decompression. As H.261 standard, MPEG video works in YCbCr color space, with downsampled chrominance components. The sampling used is 4:2:0, and the luminance resolution cannot go higher than 768x576 pixels. It makes also possible to have different resolutions, which allows a frame rate varying from 23.98 to 60 fps. There is also an improvement with respect to H.261 and H.263, since there is spatial redundancy and temporal redundancy: the first encodes a single image with the JPEG encoding, while the second diversifies the encoding for each frame.

MPEG expands H.261 and H.263 compression algorithms providing a more sophisticated scheme of motion estimation:

- **I** frames (***intra coded frame***): these frames are encoded with JPEG algorithm, independently and with low quality. Even though, they require higher memory space, but make possible the random access.
- **P** frames (***predictive coded frame***): they are encoded based on an estimation referred to the previous I or P frame. The differences are calculated starting from the absolute value of luminance components, occupying a smaller space, with the risk of error propagation.

- **B** frames (***bidirectionally predictive coded frame***): two motion estimations are used for encoding, related to the previous and following frames: a bidirectional estimation (previous and future frames). These are the most complex.



The storage sequence is the following: IPBBPBBIBB. If I lose B there is an error (which is limited there) but if I lose a P frame the error propagates to more B frames. More I-frames allow random access in more time points, with an exchange of increasing bitrate, and having so a better final quality. The schema IBBPBBPBBIBBPBBPBB... is followed, and **there must be one I-frame every 15 frames**. As for 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.

The **motion compensation and prediction algorithm** work in three phases:

1. Motion estimation of objects and motion vector creation.
2. Frames estimation using information collected in the previous phase.
3. Comparison between the estimated frame and the real one to calculate the error.

At the end of this process only the motion vector and the error estimation are saved, this way we can bring with us just the estimation which is way lighter than the actual frame.

MPEG works with a half bit precision, where each 16x16 block is expanded to a virtual copy of 32x32, the search of the new position is done via in the new macro-block, then interpolated in the original one. Expanding the macroblock, the size of the search area is ¼ of the original size. Again, as before, the high complexity comes from the research algorithm.

One of the main problems of this approach is the size of macro-blocks that need to be applied to the motion compensation prediction algorithm: blocks of bigger size create a lower prediction algorithm with a less time to encode, while smaller sizes bring an increasing complexity but return more precision. Hence, a first solution was: blocks are chosen between variable dimensions, for example, the *quad-tree* method, where **different parts of a frame are decomposed in smaller and smaller blocks in a spiral-like pattern counterclockwise**. Macroblocks are divided until they present difference from the original image. The advantage of a technique is the fact that the predictions are more accurate, meaning also that fewer and fewer differences will need to be encoded. But this is, again, very expensive computationally,

and the description of the various regions of the macro-block are really complex. That's why this solution has not been applied in real scenario.

MPEG on its own is able to encode CIF images, 352x288, at a comparable quality with respect to the original and a compress ratio of 30:1. It is possible to reach a higher compression ratio, at the price of a lower quality. Its main applications are still today video on CD (demo at most), old videogames and distance education, but not in a real-time streaming scenario.

As soon as the quality of images increased over time, new standards became necessary.

6.3 MPEG-2

Differently from its predecessor, MPEG-2 achieves high quality for DVD-ROM, with bitrates higher than 4Mb per second, its closest comparison is an old commercial television broadcasting. It is the standard format for high-quality consumer applications, but requires specific hardware for decompression, even though its wide-spread as for today. MPEG-2 supports 5 different motion prediction procedures, which are:

1. *Frame prediction for frame-picture*, the one used by MPEG-1.
2. *Field prediction for field-picture*. It is possible to divide each frame in top-field (odd lines of the frame) and bottom-field (even lines of the frame). We have four different motion vectors that predict the top-field and bottom-field for both previous and following frames. This allows to have more precision.
3. *Field prediction for frame-picture*.
4. *16x8 MC for field-pictures*. Used for fast movements. It allows to have more precision.
5. *Dual-prime for P-pictures*.

To preserve the vertical details (variations) we move from a zig zag scan to an alternate scan for images (different approach compared to JPEG). The main differences with MPEG-1 are the improved error resistance, the support of chromatic under-sampling 4:2:2 and 4:4:4 (instead of 4:2:0), a non-linear quantization and an overall higher flexibility. MPEG-2 also uses another frame: "**D** frame" (come from DC component), used during fast forward/rewind and not in normal playback → in this case we don't care about the quality of the image but just on identify the objects in the video space: just to give an idea about the content of the video.

6.4 MPEG-4

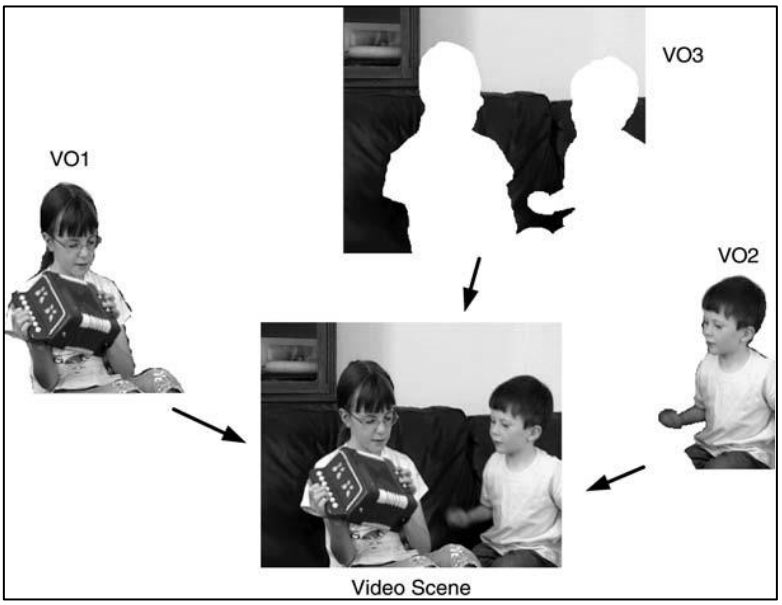
This new advancement in the MPEG family allows to integrate video streams and objects created independently, due to the possibility of indexing single elements of a given scene. It is intended for complex and interactive multimedia systems, but it is supported for different devices and bandwidths, also thanks to the optimization of three different bit-rates (< 64Kb per second, 64-384 Kb per second, 384-4Mb per second). The applications of this protocol are

varied and can adapt to modern requirements, like video streaming on the internet, videos for smartphone, virtual meetings, television production and so on.

A video in MPEG-4 can be decomposed in two parts: a background movement, limited to the camera movements, encoded hence as a fixed image, and the actual movements. This creates a hierarchical description of a scene, as such:

1. **Video-object Sequence** (VS): the complete scene, it contains both natural and synthetic objects.
2. **Video Object** (VO): a particular scene object, that can have an arbitrary shape, which can be either an object or the background itself of the scene.
3. **Video Object Layer** (VOL): each VO can have several VOL, or only one, and these components are used for scalable encoding (in the first case) or non scalable encoding.
4. **Group of Video Object Plane** (GOV): an optional level that allows considering a sequence of VOP.
5. **Video Object Plane** (VOP): a snapshot of a VO in a particular moment. The shape of each VOP is arbitrary and must be encoded together with texture (using grayscales). Each VOP is divided into 16x16 blocks, and the motion vector for the global object is calculated.

Example of VOP with MPEG4



Here we can see how the singular parts of the images were cut, leaving a mask: starting from the single element of an image, like the VO1, and adjusting the grey-scale to the borders of it, we could paste it in any different background, composing new frames.

6.5 MPEG family

MPEG-7 defines how to represent a content descriptor in a standard way, associating to objects a set of descriptors to allow classification and content search. It defines generic containers for objects of different media of different standards, with descriptions thought for human users. It is hence intended for information retrieval. Although, it does not define how to extract content descriptions and how to use those descriptions.

As for the protection of content, MPEG-21 was theorized around 2003, which allows a content description with the rights of who created them. It must provide an interface to make media usage easier.