



**Department of Economic Informatics and Cybernetics**  
Bucharest University of Economic Studies

# Multimedia

Liviu-Adrian Cotfas, PhD.



[liviu.cotfas@ase.ro](mailto:liviu.cotfas@ase.ro)

# Few words about me...



<https://ro.linkedin.com/in/cotfasliviu>

## Evaluation

- Final test – 60%
- Seminar – 40%
  - project (mandatory)

# Recommended Reading / Watching

- Slides, Examples, Books:
  - <https://github.com/liviucotfas/ASE.Multimedia>
  - <http://online.ase.ro>

Administrative issues

## Further Reading / Watching

- Courses on Microsoft Virtual Academy - [mva.microsoft.com](http://mva.microsoft.com)
  - Free
- Courses on PluralSight - [www.pluralsight.com](http://www.pluralsight.com)
  - Free trial
  - Free access (limited period) through [Microsoft DreamSpark](http://Microsoft DreamSpark)

# Definition, Characteristics, Concepts

# Definition

- **Multimedia** is:
  - any combination of **text, graphics, video, audio, and animation**
  - in a distributable format that consumers can interact with using a digital device.
- **Multimedia** can be thought of as a super-medium of sorts, because it represents the blending together of previously distinct, and noncombinable, forms of human expression

# Triggering factors

- The development of multimedia has been made possible by the **Digital Revolution**, through:
  - analog-digital conversion;
  - data compression.
- The **Digital Revolution** represents the change from mechanical and analogue electronic technology to digital electronics which began anywhere from the late 1950s to the late 1970s with the adoption and proliferation of digital computers and digital record keeping that continues to the present day.

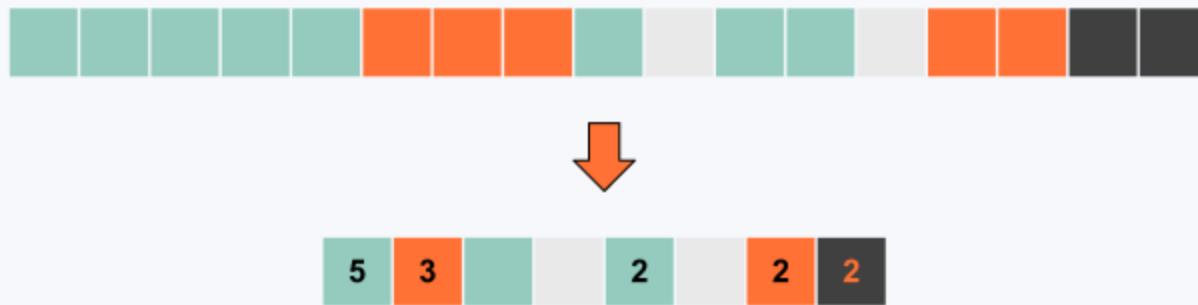
# Data Compression

- There are two major categories of compression algorithms:
  - Lossless compression
  - Lossy compression
- Compression algorithms used in multimedia are usually **asymmetrical** – the compression time is greater than the decompression time.

# Lossless compression

- substitutes a more efficient encoding to reduce the file size while preserving all of the original data. When the file is decompressed it will be identical to the original.
- Run Length Encoding - RLE
  - one of the simpler strategies to achieve lossless compression
  - can be used to compress bitmapped image files (ex: \*pcx format). Bitmapped images can easily become very large because each pixel is represented with a series of bits that provide information about its color. RLE generates a code to "flag" the beginning of a line of pixels of the same color. That color information is then recorded just once for each pixel. In effect, RLE tells the computer to repeat a color for a given number of adjacent pixels rather than repeating the same information for each pixel over and over. The RLE compressed file will be smaller, but it will retain all the original image data—it is "lossless."

# Run Length Encoding - RLE



# Lossy Compression

- the number of bits in the original file is reduced and some data is lost. Lossy compression is not an option for files consisting of text and numbers, so-called *alphanumeric* information. Losing a single letter or number could easily alter the meaning of the data.
- often possible to maintain high-quality images or sounds with less data than was originally present (especially useful for multimedia)
- exploits the limits in human perception

# Lossy Compression

- Examples:
  - JPEG – images;
  - MPEG – sound and video.
- MP3 compression:
  - analyzes the sound file and discards data that is not critical for high-quality playback;
  - removes frequencies above the range of human hearing. It may also evaluate two sounds playing at the same time and eliminate the softer sound. These types of data can be eliminated without significant impact on quality;
  - can reduce by a factor of 10 the amount of data required to represent digital audio recordings yet still sound like the original uncompressed audio to most listeners.

# Multimedia Applications

- Characteristics:
- Multimedia systems must be computer controlled.
- Multimedia systems are integrated.
- The information they handle must be represented digitally.
- The interface to the final presentation of media is usually interactive.

# Approaches for building multimedia applications

- Multimedia authoring – high-level
  - involves the assembly and bringing together of Multimedia with possibly high level graphical interface design and high level scripting
  - well-known multimedia authoring software: Animate / Flash (Adobe), Director (Adobe)
- Multimedia programming – low-level
  - assembly, construction and control of Multimedia that involves programming languages such as C, C# and Java and specialized libraries.

# Classification of multimedia applications

- Several classification criteria based on:
  - domain: economy, education, advertising, medicine, industrial, entertainment, navigation and information systems, communications;
  - interactivity: interactive, static;
  - location: local, remote (video-streaming, distant learning).

# Hardware / Software Requirements

# Hardware Requirements

- Input and processing hardware for:
  - Images;
  - Sound;
  - Video.

# Input Devices - Images

- Scanners
  - capture text or images using a light-sensing device.
  - types of scanners: flatbed, sheet fed, and handheld
  - operation: a light passes over the text or image, and the light reflects back to a **CCD** (charge-coupled device). A **CCD** is an electronic device that captures images as a set of analog voltages. The analog readings are then converted to a digital code by another device called an **ADC** (analog-to-digital converter) and transferred through the interface connection (usually USB) to RAM.
  - Optical character recognition (OCR) is the process of converting printed text to a digital file that can be edited in a word processor.

# Input Devices - Images

- scanner

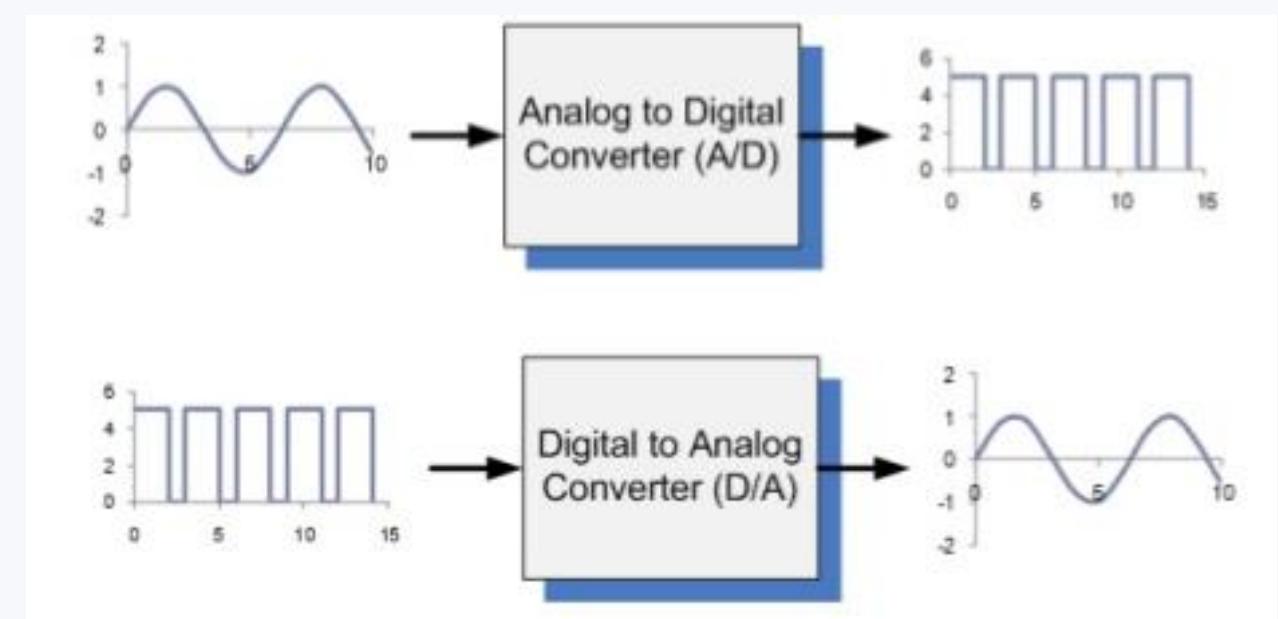
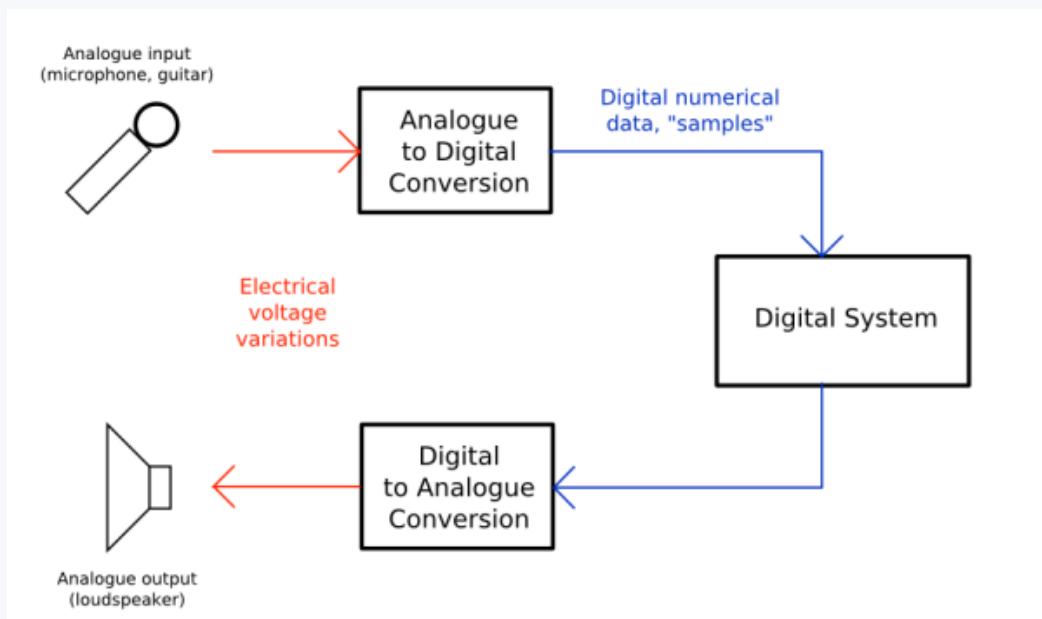


# Input Devices - Images

- **Digital cameras**
  - When the camera shutter is opened to capture an image, light passes through the camera lens. The image is focused onto a CCD, which generates an analog signal.
  - The analog signal is converted to digital form by an ADC and then sent to a digital signal processor (DSP) chip that adjusts the quality of the image and stores it in the camera's built-in memory or on a memory card.

# Input Devices - Sound

- Sound card
  - converts the analog signal from the microphone to a digital representation;
  - converts the digital representation to a analogue one that can be played back by the speakers.



## Input Devices - Video

- Video card
- Digital video camera

# Software Requirements

- **Device drivers** - computer program that operates or controls a particular type of device that is attached to a computer. A driver provides a software interface to hardware devices, enabling operating systems and other computer programs to access hardware functions without needing to know precise details of the hardware being used.
- **OS Utility Multimedia Applications** – music player, video player, image viewer, basic image editor (ex: Windows Media Player)
- **Application Software**

# Application Software

- An application is software that performs a specific task. These programs combine with the operating system to make computers productive tools.
- There are two major types of software for multimedia development:
  - **Media-specific** applications are used to create and edit the individual media elements (text, graphics, sound, video, animation) that make up a multimedia product.
  - **Authoring** applications contain software tools to integrate media components and provide a user interface.

# Graphics

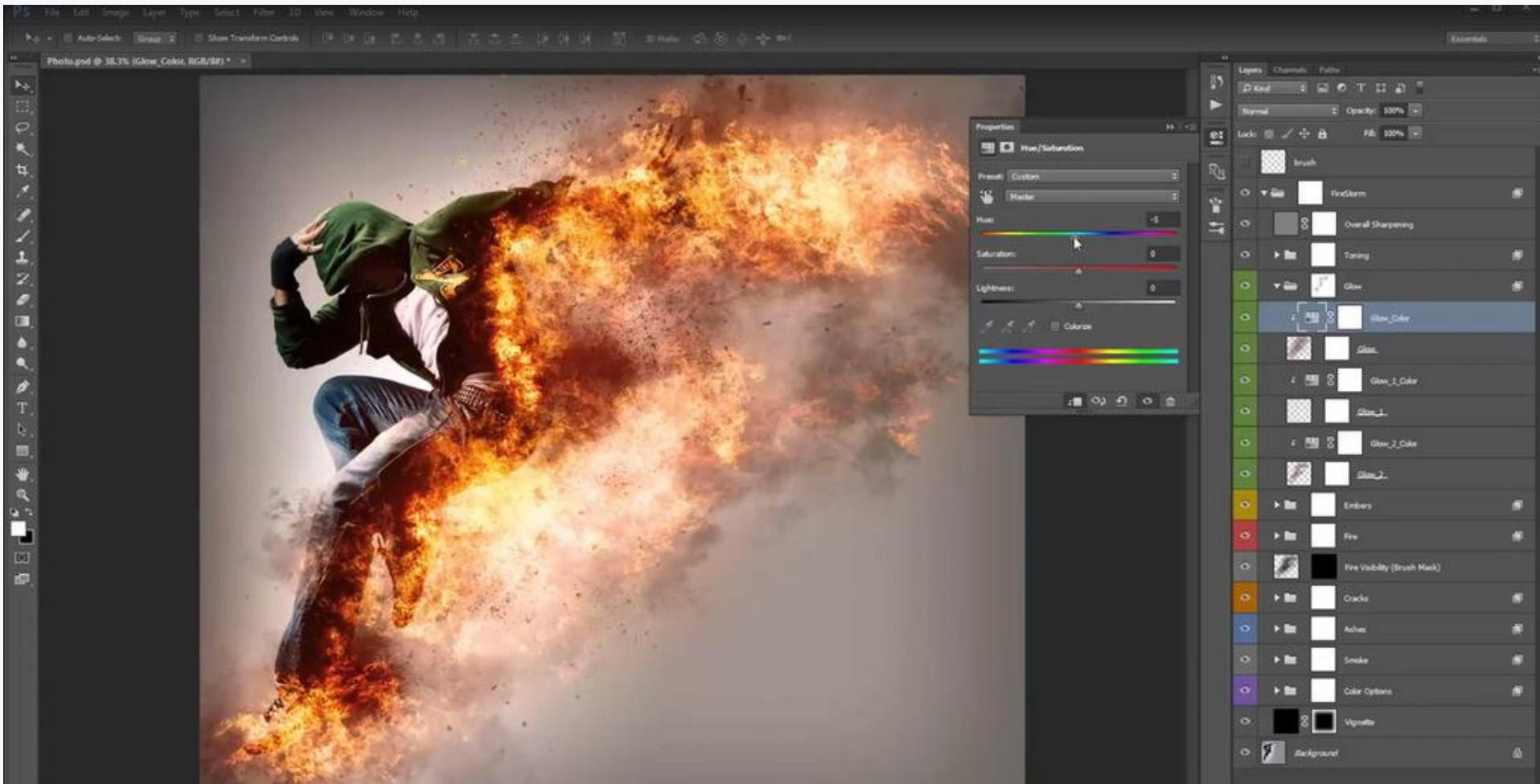
- generate 2-D or 3-D paint and draw images.
- Categories:
  - Paint programs;
  - Draw programs;
  - 3-D imaging programs.

## Paint programs

- contain tool sets to create graphics objects as well as editing tools for digital photos or scanned images
- offer a wide array of features such as filters (blur, emboss, pixelate), image adjustment settings (scale, brightness, rotate), and special effects (drop shadow, gradient overlay).
- provide special control over individual image elements is possible using layers and mask options. Text tools are used to generate graphics text with distinctive patterns, shapes, and 3-D effects.

# Paint programs

- Examples: Photoshop (Adobe), Gimp (open source)



## Draw programs

- contain a distinctive set of tools for creating basic shapes such as ovals, rectangles, Bezier curves, and polygons generated from mathematical formulas
- the shapes can be grouped, filled, and scaled to produce complex drawn images.
- such programs can be used to create unique logos, designs, and graphics objects that can easily be resized for specific multimedia projects.
- Examples: Illustrator (Adobe), Draw (Corel), Inkscape (open source)

## Draw programs

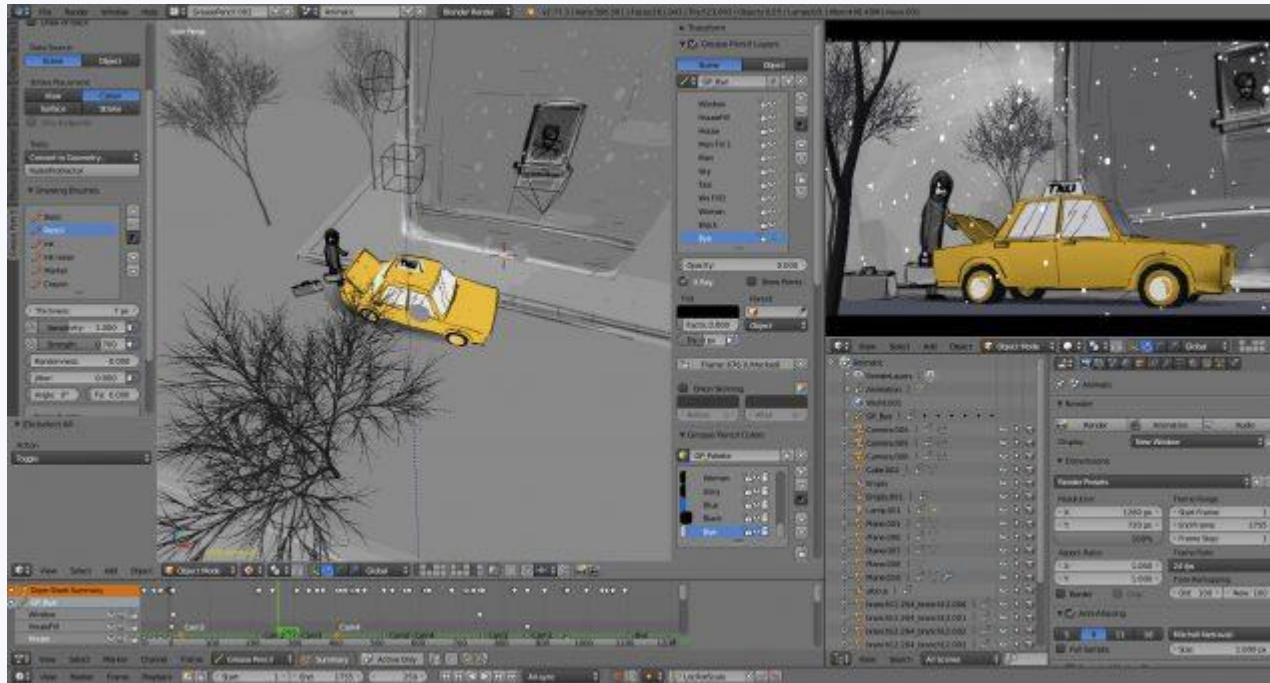


# 3-D imaging

- used to model 3-D objects, define surfaces, compose scenes, and render a completed image;
- In *modeling*, the graphic artist creates the shape of an object; in *surface definition*, color and texture are applied; in *scene composition*, objects are arranged, lighting is specified, and backgrounds and special effects are added. The final stage of 3-D graphics is *rendering*. *Rendering* creates a 3-D image from the specified scene. Rendering is both processor intensive and time consuming because the software must calculate how the image should appear based on the object's position, surface materials, lighting, and specific render options.
- Examples: Blender (open source), Bryce

# Graphics

## 3-D imaging



# Sound

- There are two major types of sound applications for multimedia development:
  - *sampled;*
  - *synthesized.*
- Examples: Audition (Adobe)

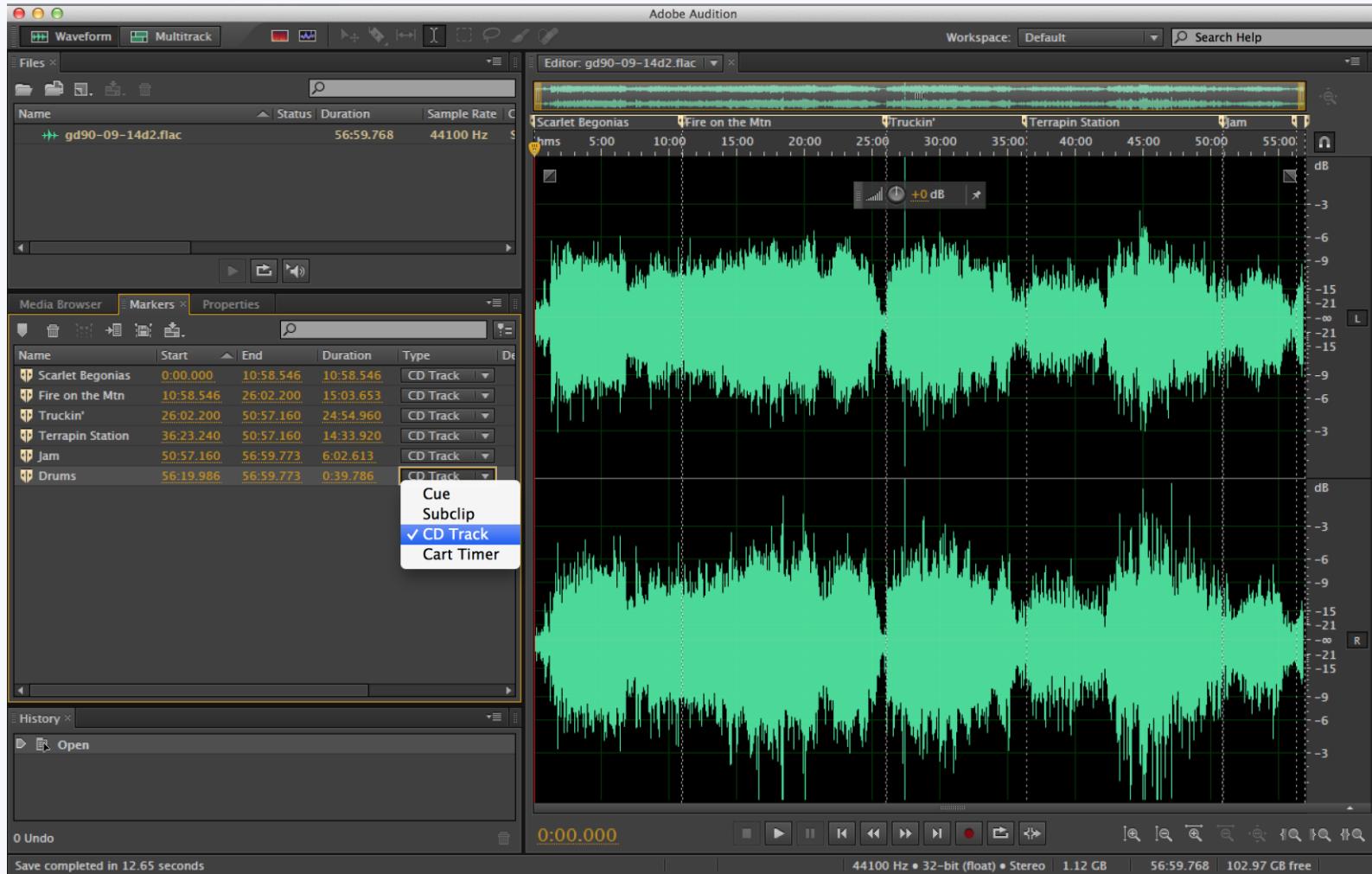
# Sampled

- sampled sounds are digital representations of analog sound sources captured from microphones or other devices. Software settings control the sound format of the sound recording.
- sampled sounds can be edited in a wide variety of ways such as trimming to delete dead space, splicing to combine sound segments, setting fade-in and fade-out (enveloping), adjusting volume, and adding special effects such as echoes or sound reversals

## Synthesized

- synthesized sound applications use digital commands to generate sounds. These commands can be captured from a MIDI instrument such as an electronic keyboard or created with a sequencer program
- the musical file is then saved and played back on a computer's synthesizer, an electronic device to generate sound. MIDI applications are a good source of original music for multimedia applications.

# Media-specific Software Sound



# Video

- an environment to combine source material called *clips*, synchronize clips to a sound track, add special effects, and save the work as a digital video.
- a video project starts by assembling film clips in a project window. The clips can be still images, animations, sounds, or digital video files.
- video applications provide tools to move and insert clips on a timeline, trim the clip, and define transitions between tracks. Sound tracks, title fields, and special effects such as superimposing, transparency, and lens flare add to the video composition. Video editing applications also define playback size and frame rate. When the video project is complete, the application provides settings to save it in specific file formats and compression schemes.

# Media-specific Software

## Video

- Examples: Premiere (Adobe)



## Animation

- used to create and edit animated sequences. **Animation** is the technique of using a series of rapidly displayed still images to produce the appearance of motion.
- Objects are drawn or imported into the software where they are manipulated in a series of still frames. Frames are played back in sequence to create an illusion of motion.
- Each frame represents a single instance of the animated sequence. Typical animation tools control the path of an object, object shape, and color changes over the frame sequence. Objects are placed on a timeline where effects can be applied to fade in, morph, rotate, spin, flip, or change pace. Multiple objects can be layered to interact with each other to create more complex animations.

## Animation

- Examples: Director (Adobe), Flash (Adobe), Animate (Adobe)

# Authoring Software

- consists of programs specifically designed to facilitate the creation of multimedia products.
- they are used to assemble media elements, synchronize content, design the user interface, and provide user interactivity.

# Authoring Software

- Categories based on the metaphor they use to organize media element:
  - card-based metaphor;
  - timeline;
  - flow diagram.

## Card-based metaphor

- elements are arranged much as they might be on index cards or the pages of a book.
- such applications are easy to use and are ideal for products such as information kiosks, lectures, and tutorials that do not require precise synchronization of individual media elements.
- Examples: **PowerPoint**, ToolBook

## Timeline

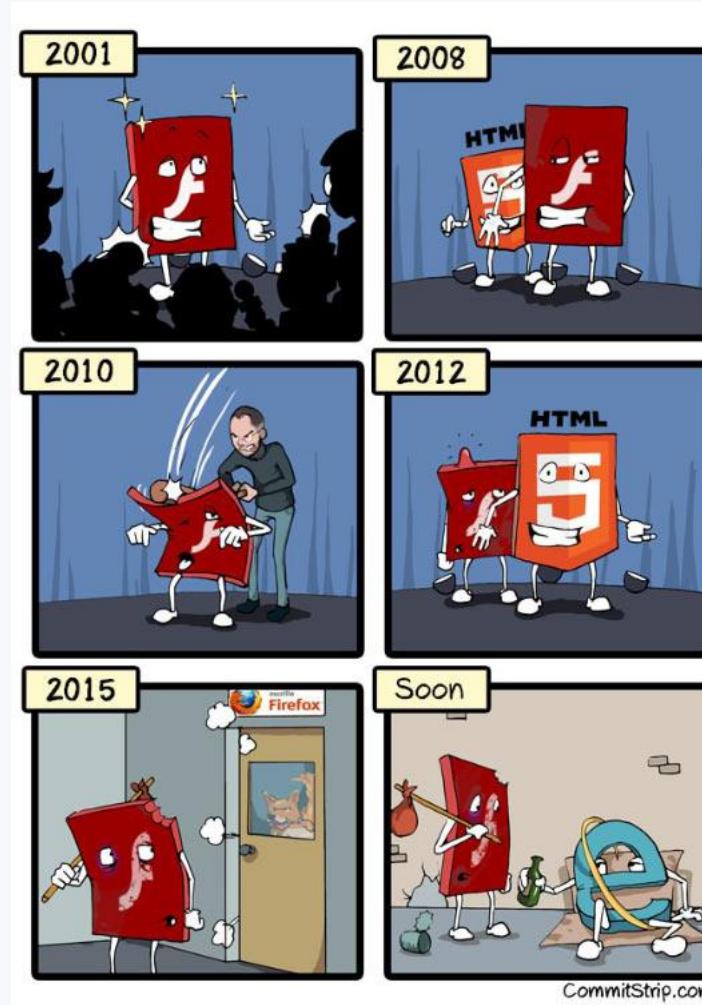
- use a *timeline* of separate frames much like a motion picture film;
- such applications provide the precise control needed for advanced animations;
- Examples: Director (Adobe), Flash (Adobe), Animate(Adobe).

# Authoring Software

# Adobe Flash



# The Evolution Of Flash



## Flow diagram

- use *icons* arranged on flow lines to quickly develop a wide range of multimedia products including advanced tutorials, product demonstrations, and simulations. Icons can represent both content (images, text, animations, video) and a wide range of interactions (play, stop, go to, calculate, etc.).
- example: Authorware

# Web Multimedia

# Web Multimedia

- HTML5 provides a greatly improved support for web multimedia, by including elements such as audio, video, canvas
- supported multimedia elements:
  - text
  - images
  - animations
  - raster graphics
  - vector graphics
  - 3D graphics
  - audio
  - video

# HTML5 Games



[http://www.cuttherope.net/basic\\_standalone/game.html](http://www.cuttherope.net/basic_standalone/game.html)

# HTML5 Videos

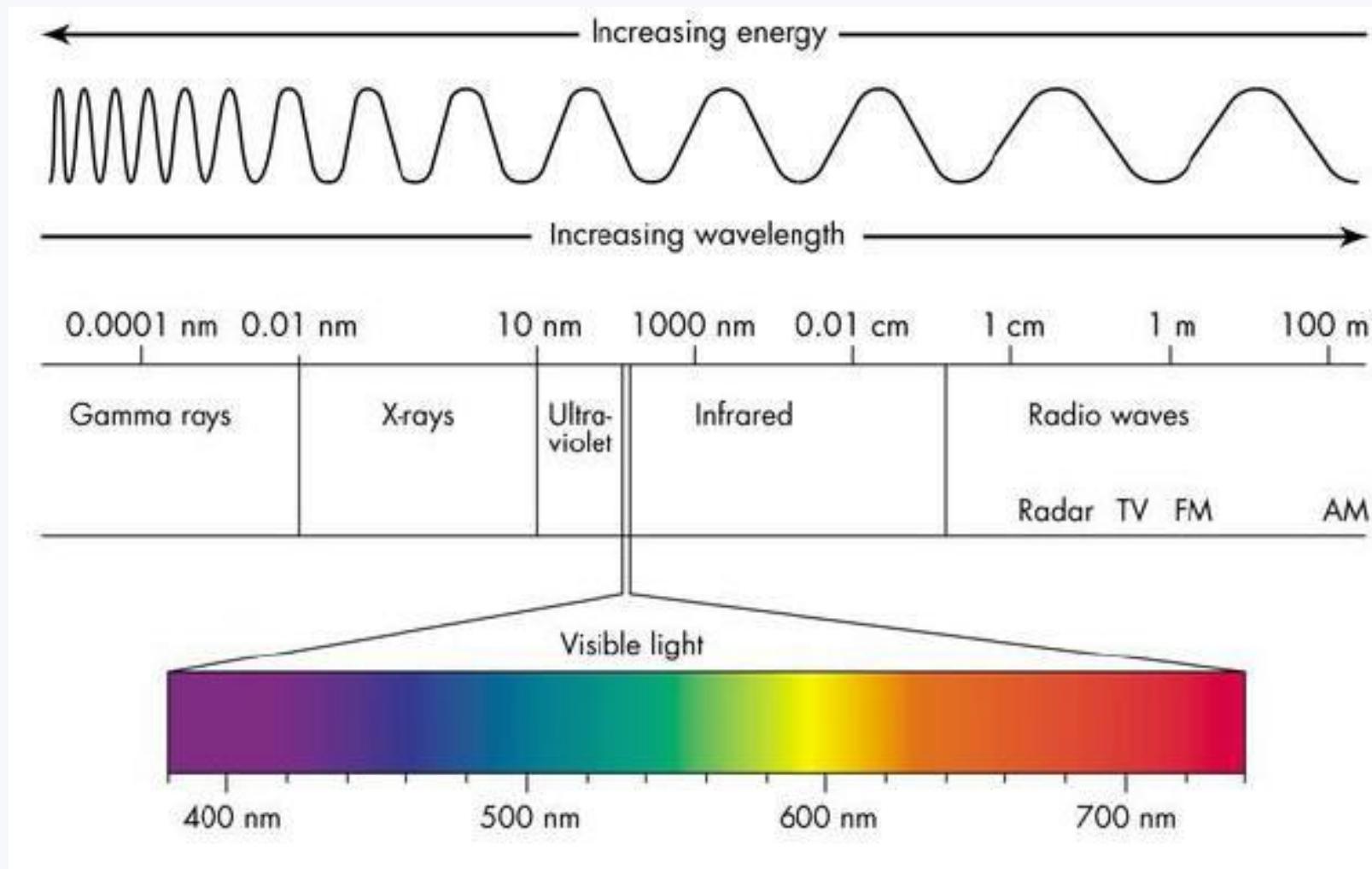


# Images

# Digital Images

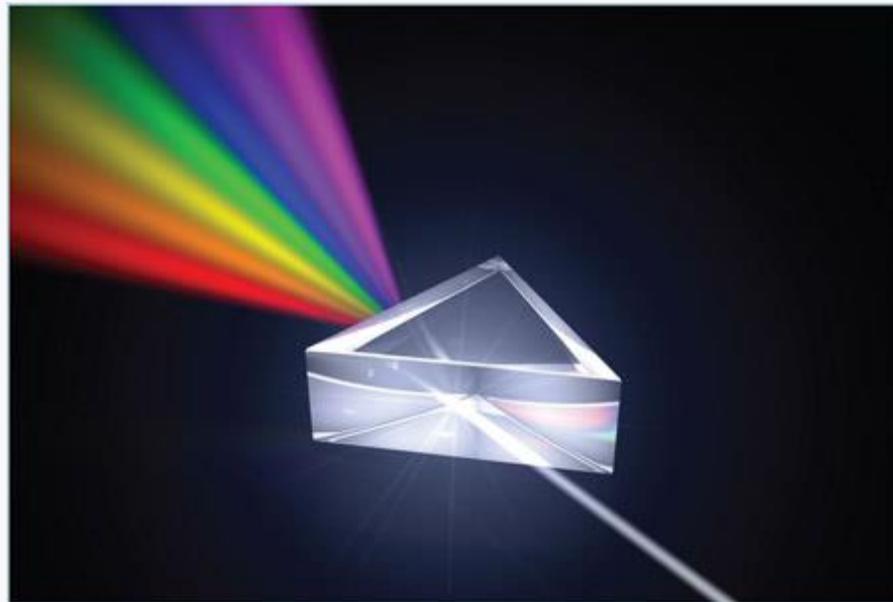
- An **image** is a two- or three-dimensional representation of a person, animal, object, or scene in the natural world [1].
- A **digital image** is a numeric representation of a two-dimensional image.

# Visible light



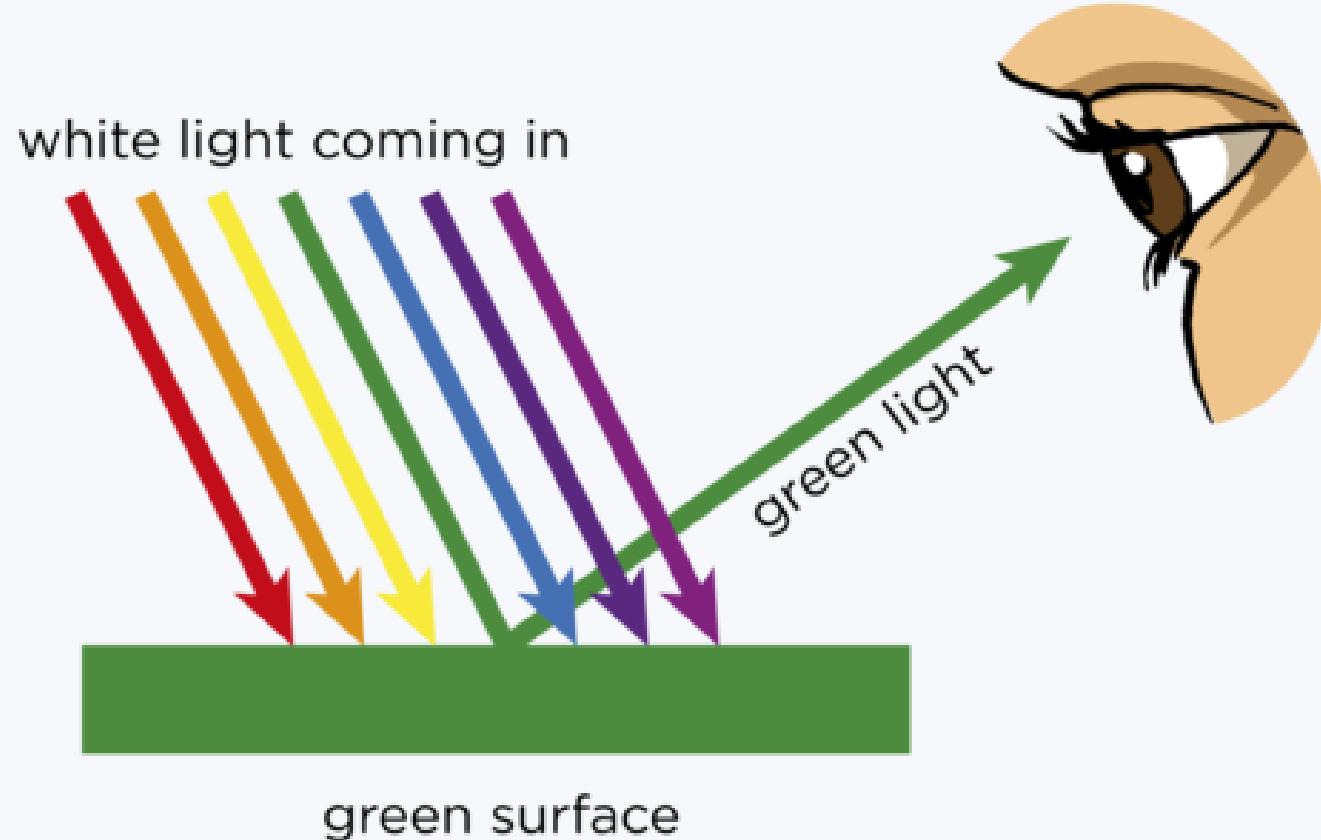
# Color Space

- we refer to natural sunlight as white light because it appears to the human eye to be colorless.
- it's possible to separate white light into a dazzling display of various colors using a prism. As light travels through a prism, it's refracted (bent), causing the beam of white light to break apart into its component color wavelengths



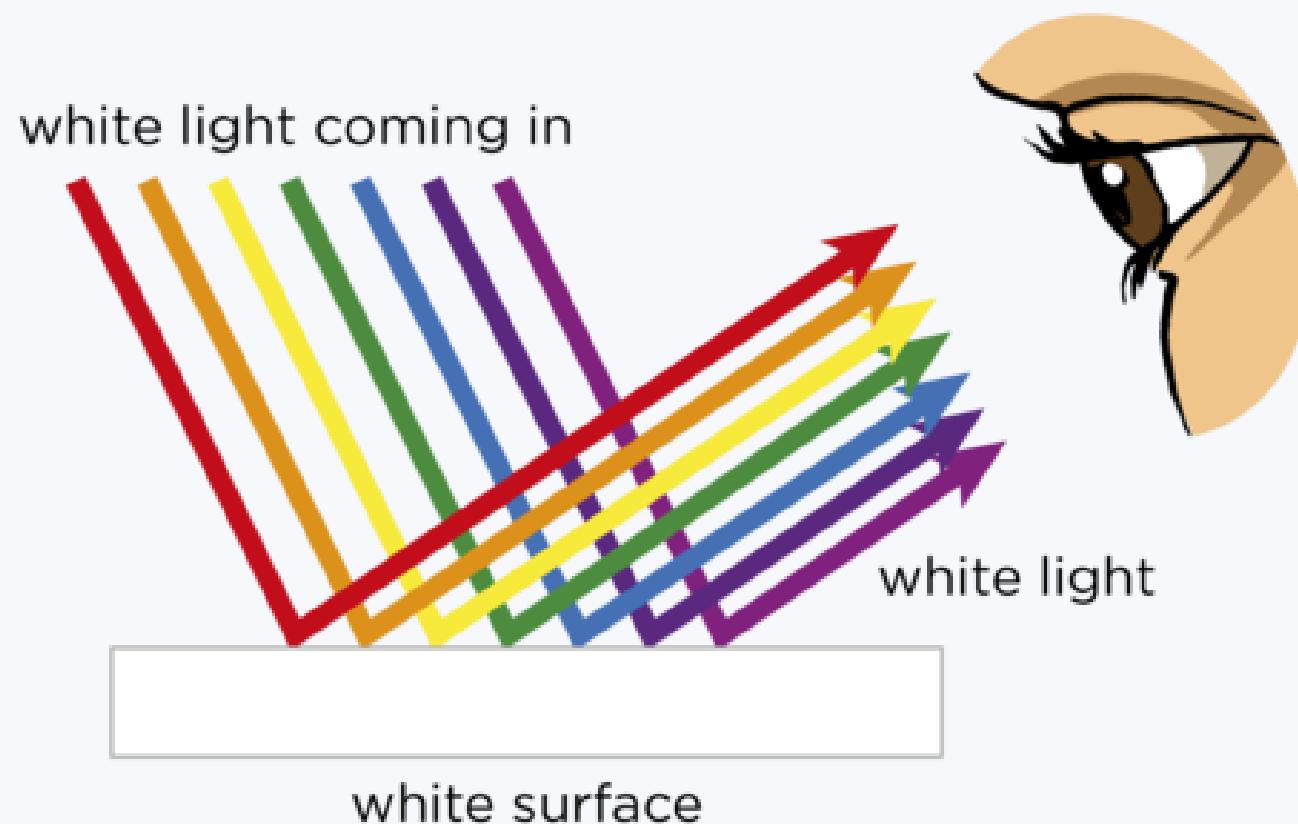
The primary and secondary colors of white light become visible when refracted by a glass prism.

# Color Space



The green leaf absorbs all the colors except green which it reflects back into our eyes.

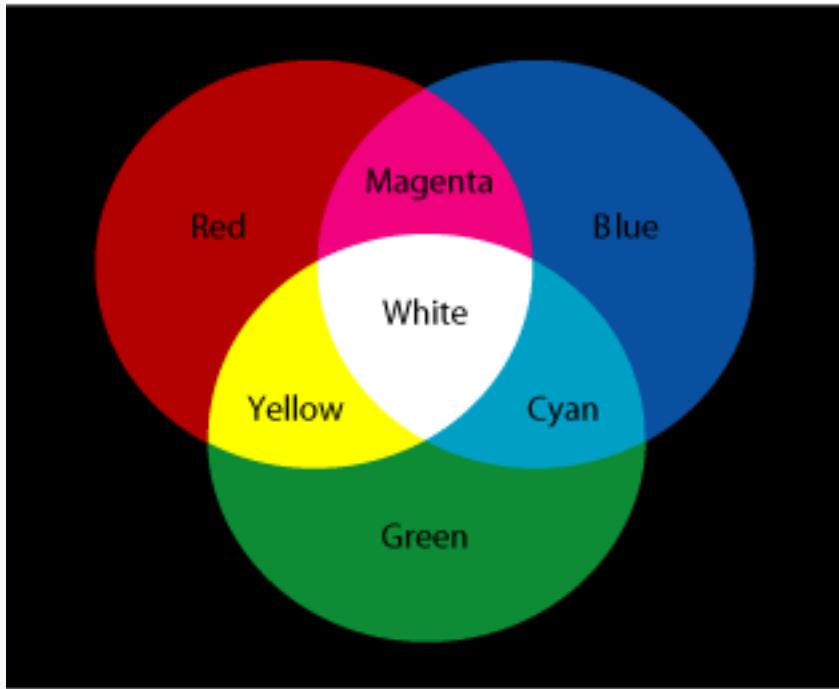
# Color Space



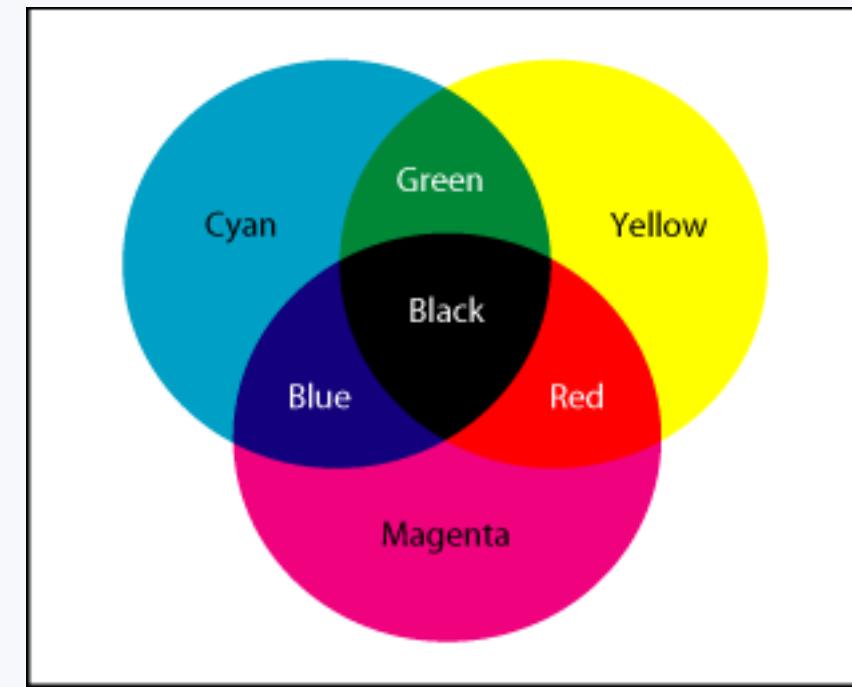
## Color Space

- Although we can get black paint as a pigment, black is not a color of light. Black is the result of the complete absorption of light.

# Color Systems



Additive Model (RGB)



Subtractive Model (CMYK)

## Additive Model

- Additive color is color created by mixing a number of different light colors.
- Shades of **red**, **green**, and **blue** are the most common primary colors used in additive color system.
- Additive color mixing begins with **black** and ends with white; as more color is added, the result is lighter and tends to **white**.
- The combination of **two** of the standard **three** additive primary colors in equal proportions produces an additive secondary color - **cyan**, **magenta** or **yellow**.

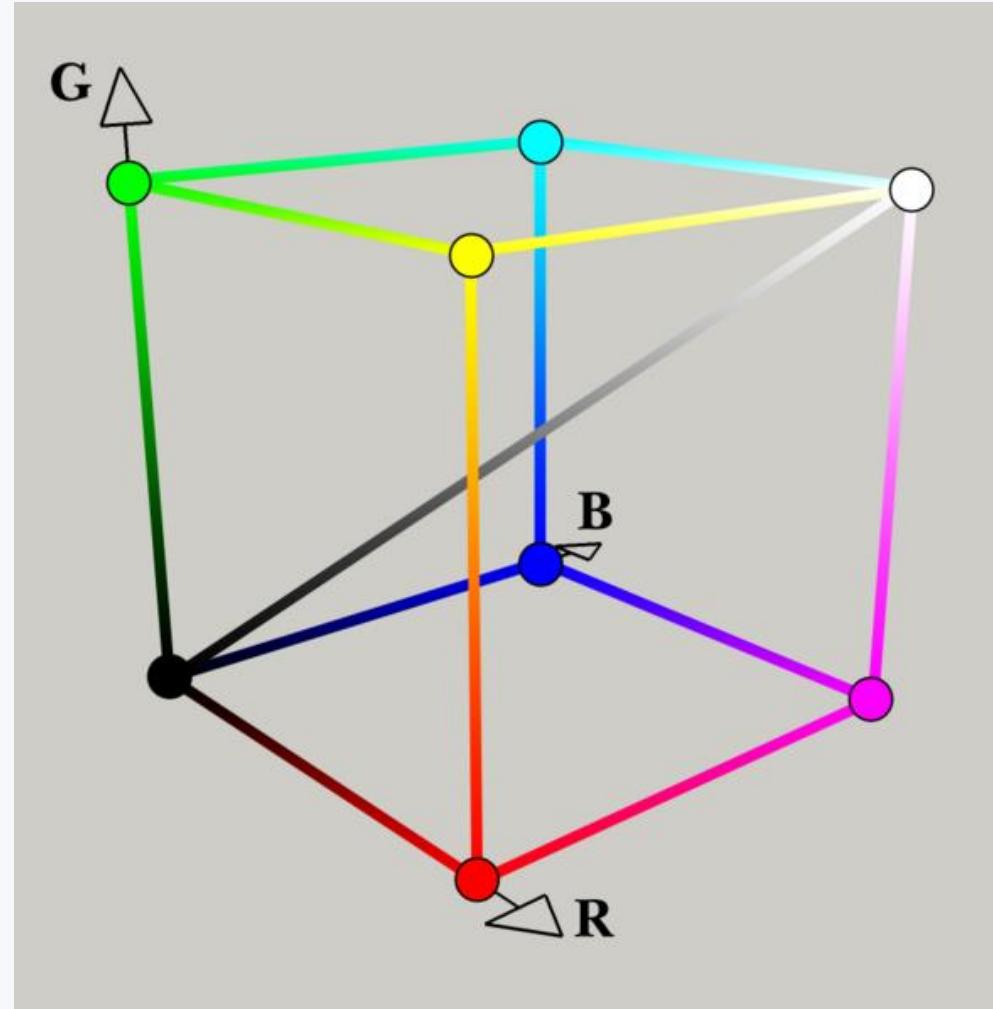
## RGB Color Model (or Mode)

- the primary colors of light are **red**, **green**, and **blue** (RGB).
- by adjusting the intensity of each, you can produce **all the colors** in the visible light spectrum. You get **white** if you add all the colors equally and **black** by removing all color entirely from the mix.
- If we were to look at a monitor such as an LCD (liquid crystal display) under a microscope, we would see that each pixel really displays only the three primary colors of light.

## RGB Color Model (or Mode)

- In additive color mixing, red and green make yellow. If you fill a graphic with intense yellow in Photoshop, the pixels really display stripes of intense red and green, with no blue.
- The individual points of color are tiny, so our brains add the colors together into yellow.

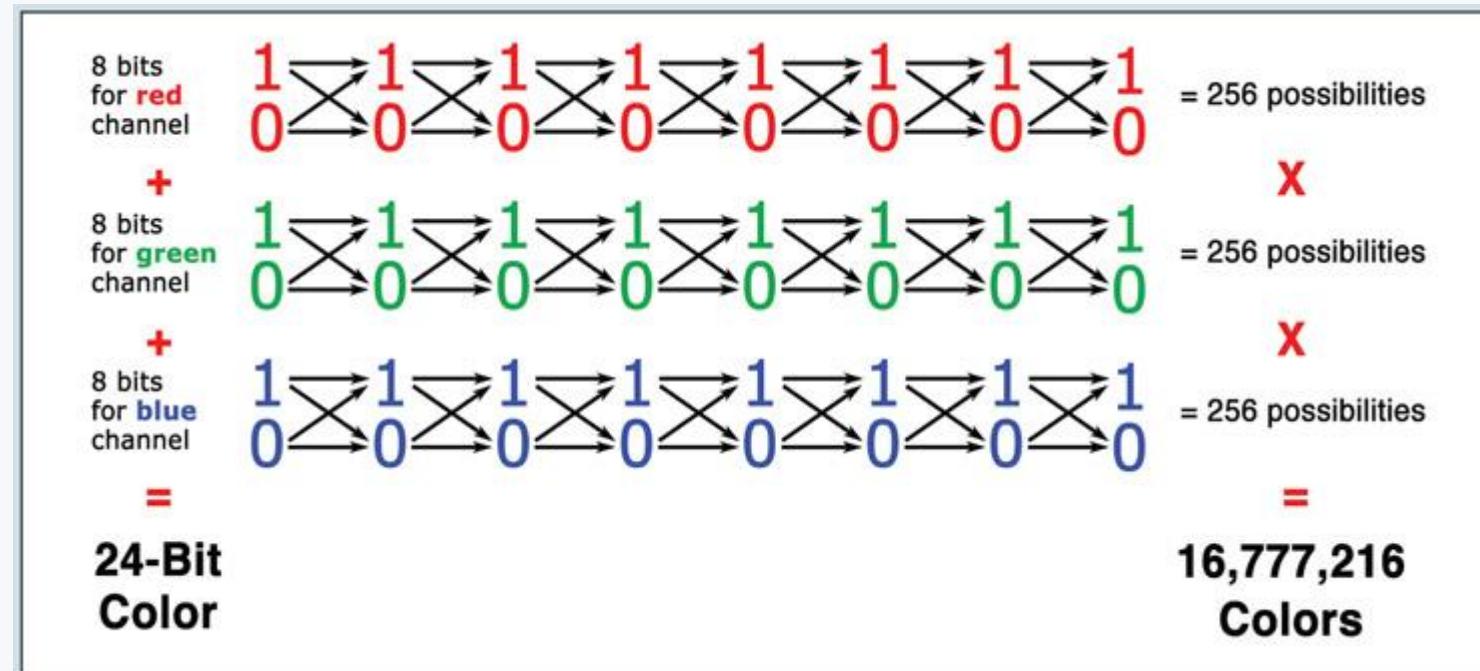
# RGB Color Model (or Mode)



# Adobe Photoshop – RGB Color



# RGB Color Model (or Mode)



The possible color combinations for any pixel in an eight-bit graphic (or a 24-bit display).

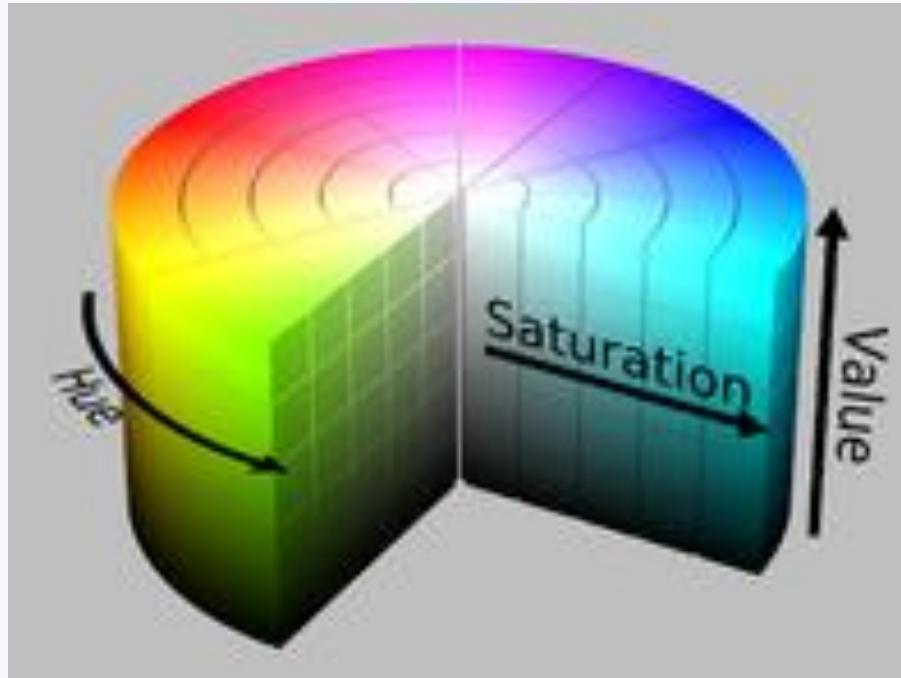
## RGB Color Model (or Mode)

- 256 possibilities for each color channel.
- 256 possibilities for red  $\times$  256 for green  $\times$  256 for blue - **16.8 million** possible combinations / colors.

# HSB - Hue, Saturation and Brightness Color Model

- Also known as **HSV** (Hue, Saturation and Value):
- Together with **HSL** (Hue, Saturation and Lightness) are the most common cylindrical-coordinate representations of points in an RGB color model.
- The two representations rearrange the geometry of RGB in an attempt to be more intuitive and perceptually relevant than the cartesian (cube) representation.

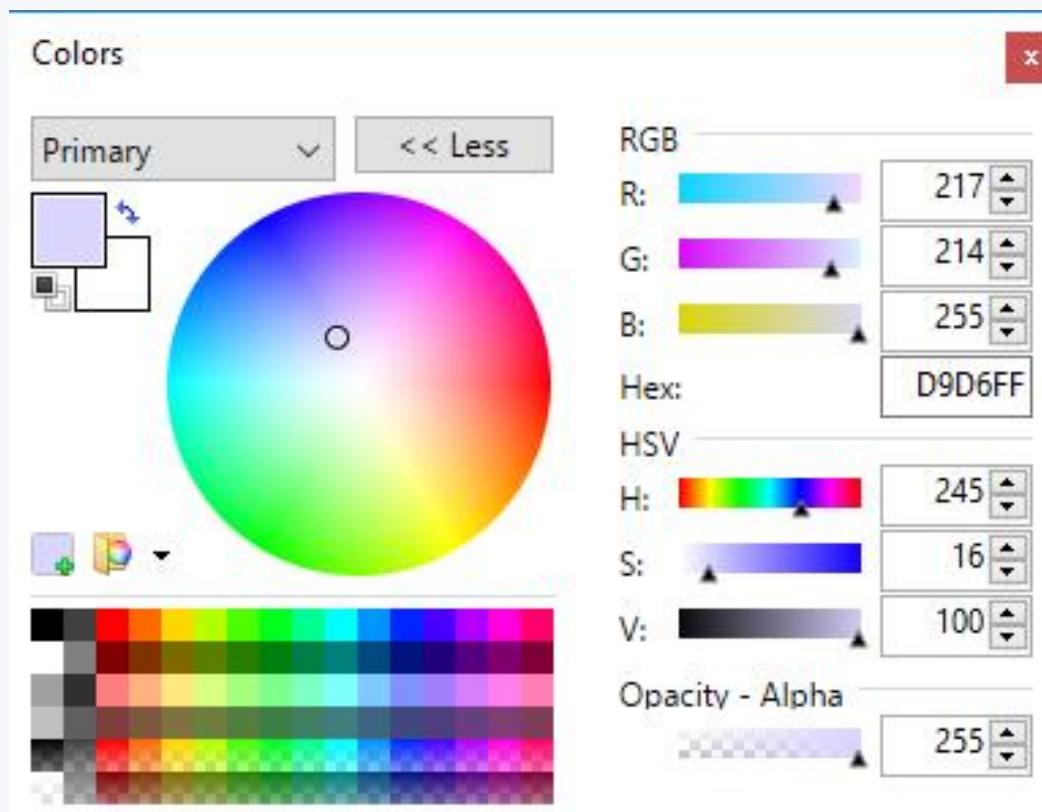
# HSB - Hue, Saturation and Brightness Color Model



- Red -  $0^0$
- Green -  $120^0$
- Blue -  $240^0$

# HSB - Hue, Saturation and Brightness Color Model

- HSL and HSV are used primarily in color pickers, in image editing software, and less commonly in image analysis and computer vision.



Color Picker in Paint.NET:  
<http://www.getpaint.net/index.html>

# Subtractive Color System

- When we mix colors using paint, or through the printing process, we are using the **subtractive** color method [1].
- A **subtractive color** model explains the mixing of a limited set of dyes, inks, paint pigments or natural colorants to create a wider range of colors, each the result of partially or completely subtracting (that is, absorbing) some wavelengths of light and not others [1]
- Subtractive color mixing means that one begins with white and ends with black; as one adds color, the result gets darker and tends to black [2].

[1] [https://en.wikipedia.org/wiki/Subtractive\\_color](https://en.wikipedia.org/wiki/Subtractive_color)

[2] [http://www.worqx.com/color/color\\_systems.htm](http://www.worqx.com/color/color_systems.htm)

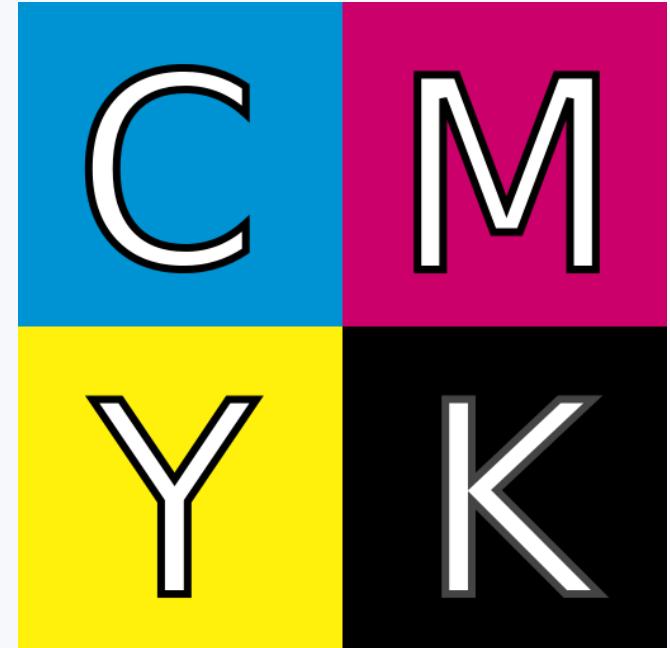
## Subtractive Color System

- The color that a surface displays depends on which parts of the visible spectrum are not absorbed and therefore remain visible [1].

[1] [https://en.wikipedia.org/wiki/Subtractive\\_color](https://en.wikipedia.org/wiki/Subtractive_color)

# CMYK Color Model (or Mode)

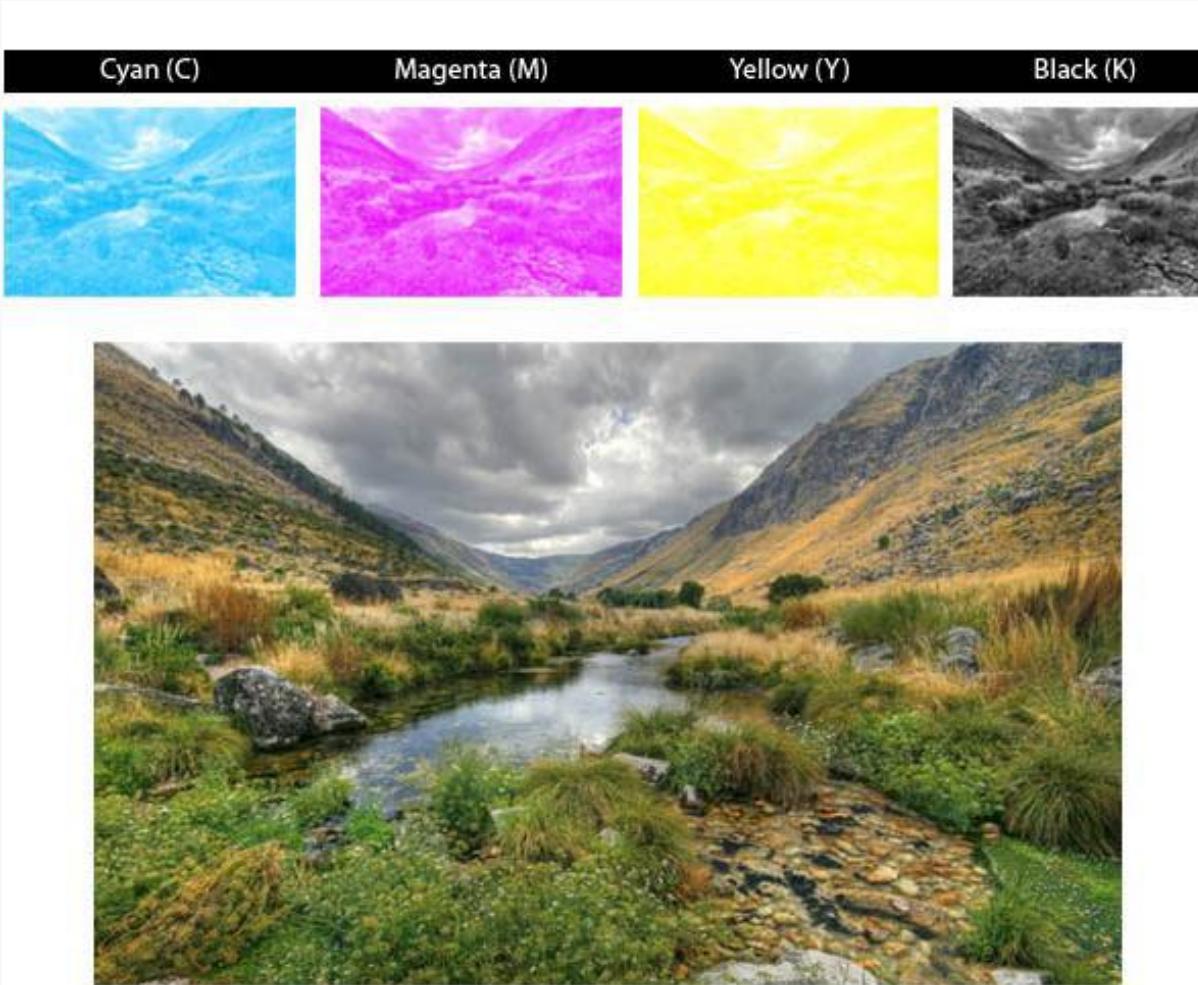
- The CMYK color model (process color, four color) is a subtractive color model, used in color printing, and is also used to describe the printing process itself [1].
- CMYK refers to the four inks used in some color printing: cyan, magenta, yellow and key (black) [1].
- Though it varies by print house, press operator, press manufacturer, and press run, ink is typically applied in the order of the abbreviation [1].



# CMY Color Model (or Mode)

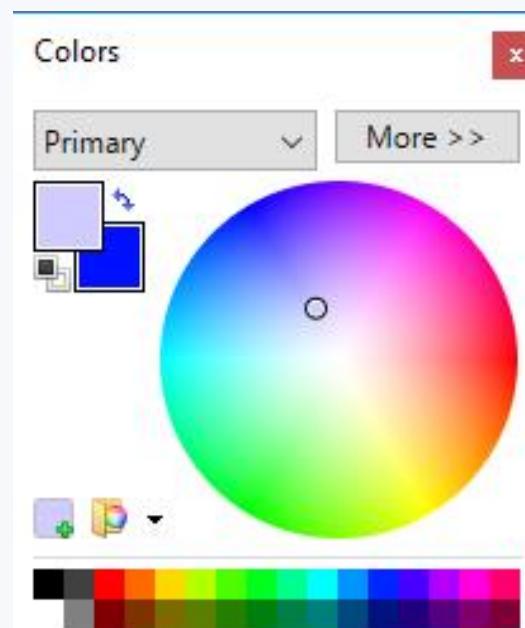


# CMYK Color Model (or Mode)



# Color Wheel

- A **color wheel** (also referred to as a color circle) is a visual representation of colors arranged according to their **chromatic relationship**. Begin a color wheel by positioning **primary** hues equidistant from one another, then create a bridge between primaries using **secondary** and **tertiary colors** [1].



Color Picker in Paint.NET:  
<http://www.getpaint.net/index.html>

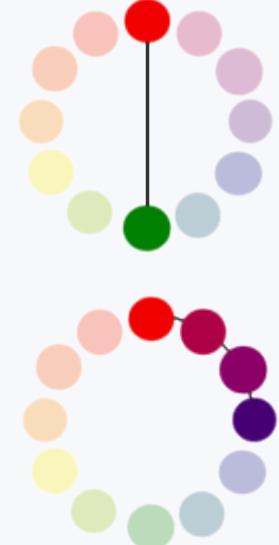
# Color Wheel

- **Primary Colors:** Colors at their basic essence; those colors that cannot be created by mixing others [1].
- **Secondary Colors:** Those colors achieved by a mixture of two primaries [1].
- **Tertiary Colors:** Those colors achieved by a mixture of primary and secondary hues [1].



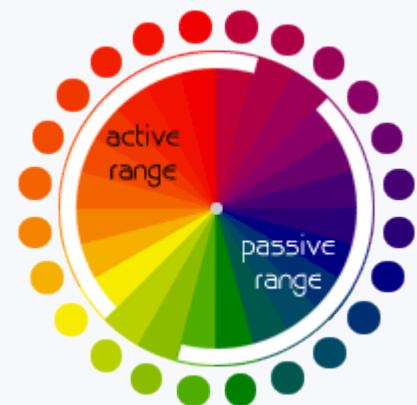
# Color Wheel

- **Complementary Colors:** Those colors located opposite each other on a color wheel [1].
- **Analogous Colors:** Those colors located close together on a color wheel [1].



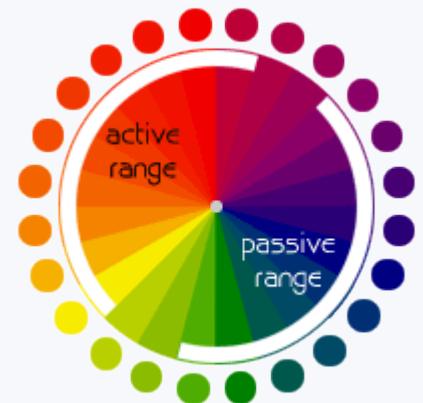
# Active & Passive Colors

- The color wheel can be divided into ranges that are visually:
  - Active colors - will appear to advance when placed against passive hues.
  - Passive colors - appear to recede when positioned against active hues.
- Advancing hues are most often thought to have less visual weight than the receding hues.
- Most often warm, saturated, light value hues are "active" and visually advance.



# Active & Passive Colors

- Cool, low saturated, dark value hues are "passive" and visually recede.
- Tints or hues with a low saturation appear lighter than shades or highly saturated colors.
- Some colors remain visually neutral or indifferent.



# Complementary Colors

- When fully saturated complements are brought together, interesting effects are noticeable. This may be a desirable illusion, or a problem if creating visuals that are to be read.



Does this text have  
highlighted edges?

- Notice the illusion of highlighted edges and raised text. This may occur when opposing colors are brought together.

# 2-D Computer Graphics

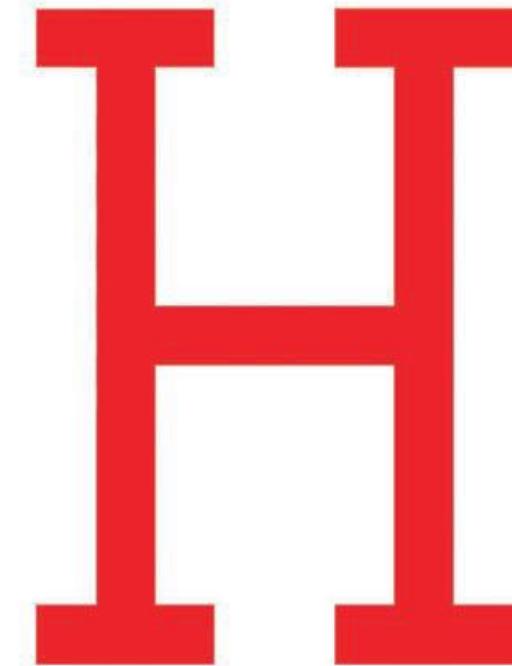
- Computers create either 2-D (width and height) or 3-D images (width, height, and depth).
- There are two main types of 2-D computer graphics:
  - **bitmapped** approach - particularly well suited for images with fine detail, such as paintings or photographs. Also called **raster graphics**.
  - **vector** drawing - used for graphic designs ranging from simple drawings and logos to sophisticated artistic creations.

# Raster Graphics

# Raster Graphics

- a **raster graphics** image is formed by dividing the area of an image into a rectangular matrix of rows and columns comprised of pixels [1].
- A **pixel**, short for picture element, is a square (or occasionally rectangular) area of light representing a single point in a raster image [2].

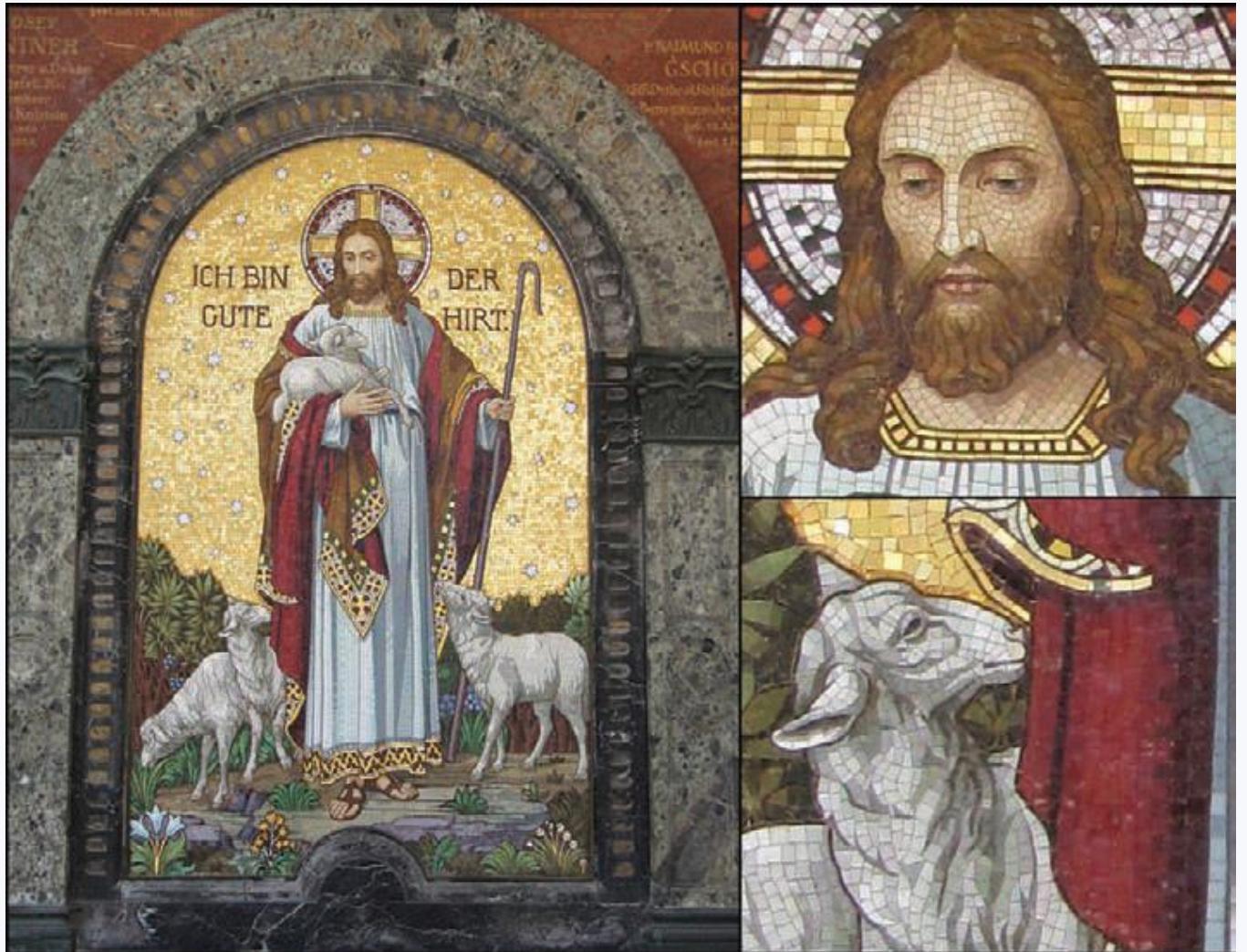
0	0	0	0	0	0	0	0	0	0	0
0	1	1	1	1	0	0	1	1	1	0
0	0	1	0	0	0	0	1	0	0	0
0	0	1	0	0	0	0	1	0	0	0
0	0	1	0	0	0	0	1	0	0	0
0	0	1	0	0	0	0	1	0	0	0
0	0	1	0	0	0	0	1	0	0	0
0	0	1	1	1	1	1	1	0	0	0
0	0	1	0	0	0	0	1	0	0	0
0	0	1	0	0	0	0	1	0	0	0
0	1	1	1	1	0	0	1	1	1	0
0	0	0	0	0	0	0	0	0	0	0



# Raster Graphics

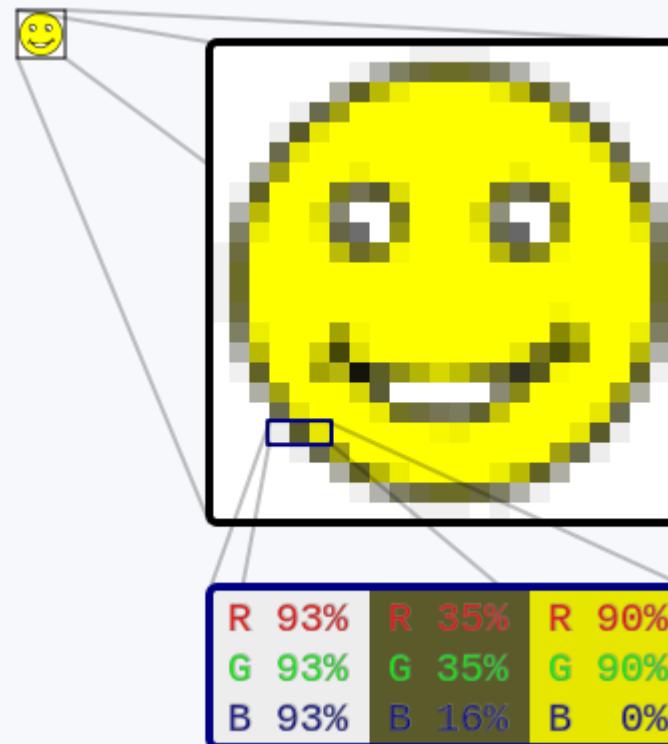
- Every pixel in a raster image is exactly the same size and contains a single color value that's typically stored as a 24-bit string of binary data.
- The total number of pixels in a raster image is fixed. In order to make a raster image physically larger, more pixels have to be added to the raster matrix.
- Likewise, pixels need to be discarded when making a raster image smaller. The width and height of a raster image is determined by how many pixels each row and column contains.

# Raster Graphics Mosaic



From a distance, the individual tiles forming the image are barely perceptible to the naked eye. Up close, however, we can see that many small pieces of visual information went into forming this 19th-century mosaic of Christ, the good shepherd. This technique of using tiny bits of colored material to form a composite visual impression dates back to about 3,000 bc.

# Raster Graphics



When enlarged, individual pixels appear as squares. Zooming in further, they can be analyzed, with their colors constructed by adding the values for red, green and blue.

## Formats

- **BMP** (Microsoft Windows Bitmap) - **\*.bmp**
  - also known as bitmap image file or device independent bitmap (DIB) file format or simply a bitmap
- either uncompressed or compressed using RLE (**lossless compression format**)
- monochrome or color (various color depths are supported)

# Characteristics

- **Resolution**
  - describes the image quality of a raster image and directly relates to the size and quantity of the pixels the image contains.
- **Pixel Dimensions**
  - describe the size of a raster image, expressed as the number of pixels along the x-axis (width) by the number of pixels along the y-axis (height).
  - ex: 800 px \* 600 px
- **Pixel Count**
  - Pixel count is the total number of pixels in a raster matrix. To determine the pixel count, multiply the horizontal and vertical pixel dimensions.
  - ex: a 30 \* 18 pixel image has a pixel count of 540 pixels.

# Characteristics

- **Pixel Density**
  - We express the pixel density or resolution of a raster image in pixels per inch (ppi)—this is pixels per linear inch (across or down), not square inch.
  - Most television and computer monitors have a display resolution of either 72 or 96 ppi.
  - The resolution determines the maximum size of an image you print. In order to produce a high-quality print of any size, digital photographs need a pixel density of at least 300 ppi—that's 90,000 pixels in a square inch!

# Characteristics

- **Color-Depth (Color Resolution)**

- measure of the number of different colors that can be represented by an individual pixel. The number of bits assigned to each pixel, or bit depth, determines color resolution.
- 8-bits (256 colors), 24 bits (16.8 million colors)
- A drawing using 16 distinct colors, for example, would only require 4 bits per pixel. Using a code with greater bit depth produces a larger file with no increase in quality.

# Advantages and Disadvantages

- Advantages:
  - great when creating rich and detailed images. Every pixel in a raster image can be a different color therefore complex images with any kind of color changes and variations can be created.

# Advantages and Disadvantages

- Disadvantages:
  - the files are often **quite large** because they contain all the information for every single pixel of the image.
  - cannot be scaled up in size very well. If enlarged, a raster image will look grainy and distorted because raster images are created with a finite number of pixels. When you increase the size of a raster image, the image increases in size however, because there are no longer enough pixels to fill in this larger space, gaps are created between the pixels in the image. The photo editing software will try to fill these gaps the best they can however, the resulting image is often blurry.
  - no information is provided regarding the shapes that make up the image.

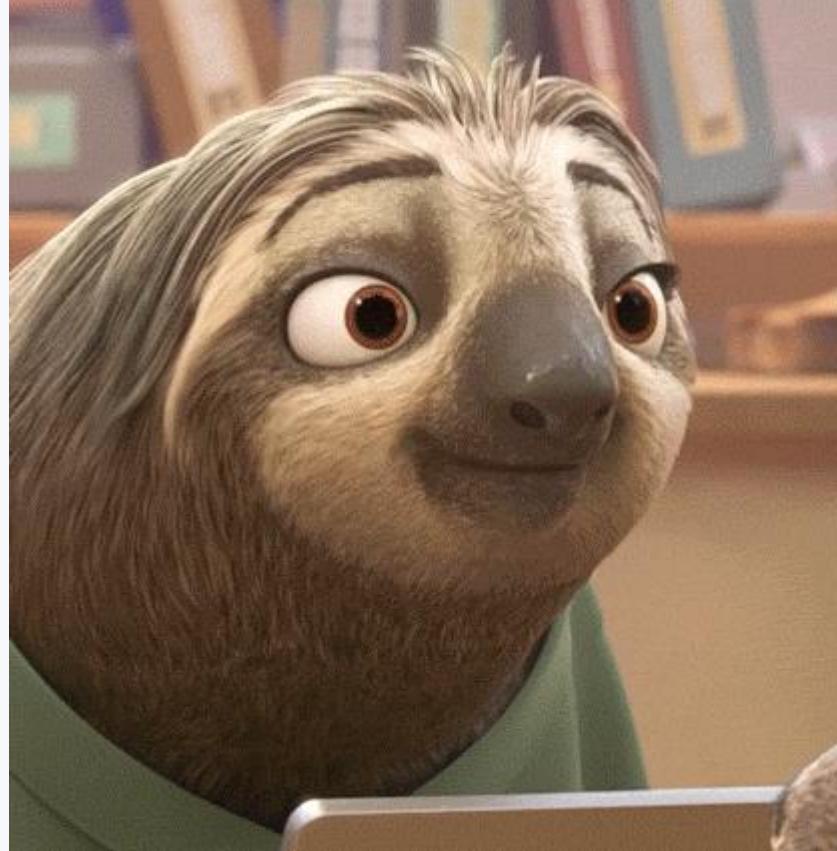
## Formats

- while working with an image file, it should be saved it in a format that supports layers (\*.PSD, \*TIFF), so it can be changed it in the future or resaved in a new format for another purpose.
- when you prepare a raster image to incorporate into a multimedia project, you'll need a **flattened, compressed** image. For lossy formats there is always a tradeoff between image quality and the file size.

# Formats

- GIF (Graphics Interchange Format) - \*.gif
  - maximum 256 colors and transparency (transparent pixels);
  - lossless compression format.
- commonly used for logos and other images with lines and solid blocks of color.
- supports interlacing, so every odd line of pixels loads, then every even line loads, making graphics seem to appear faster (users see the full-sized, half-loaded graphic before the rest of the pixels appear).
- supports animation.

# 2-D Computer Graphics Formats



<https://dl.dropboxusercontent.com/u/70618257/HTML5Multimedia/giphy.gif>

## Formats

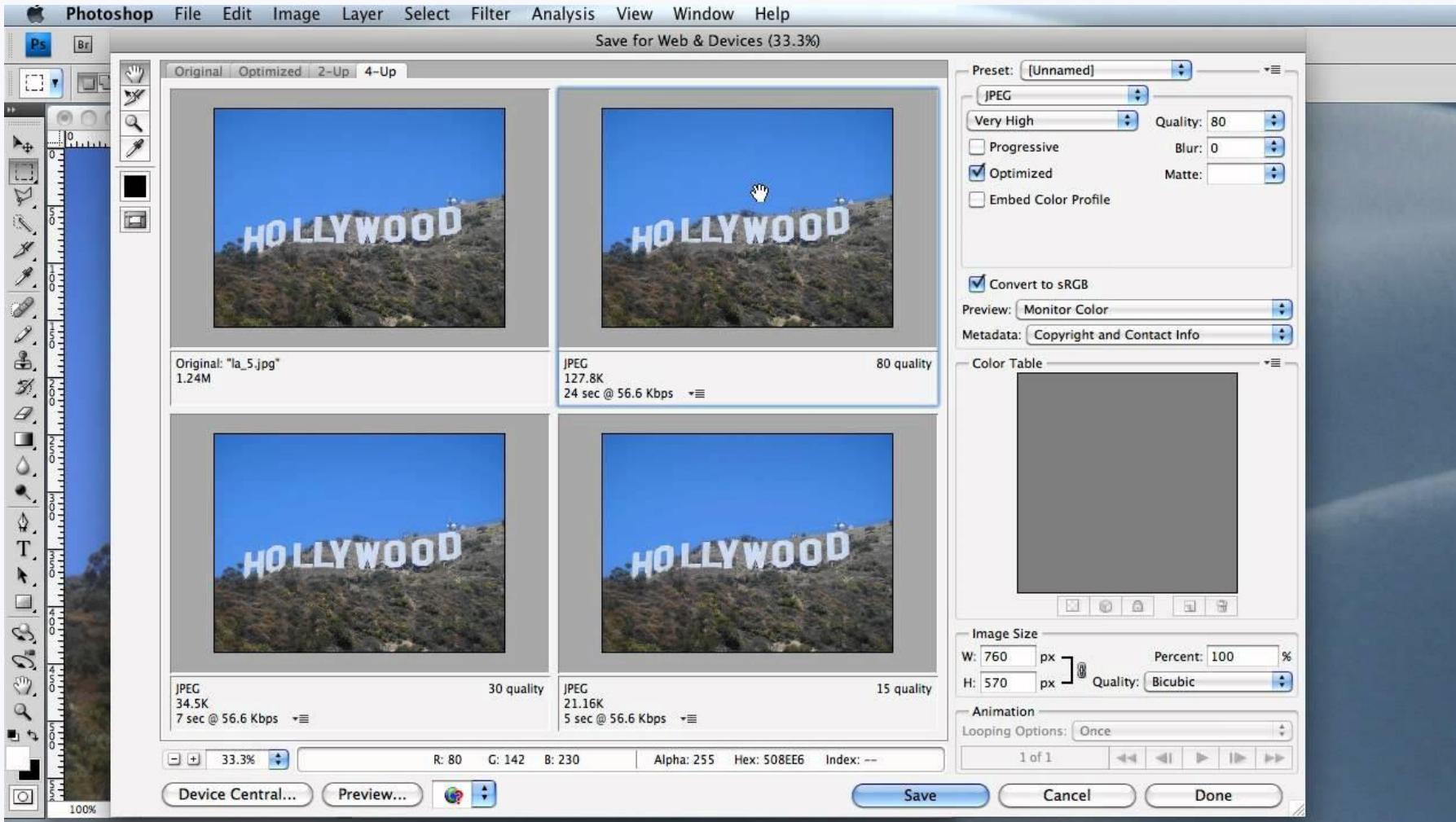
- **JPEG** (Joint Photographic Experts Group) - **\*.jpeg**
  - offers 16.8 million colors
  - does not support transparency (has no transparent pixels).
  - lossy compression format and is used most often for photographs.
  - does not support interlacing.

# 2-D Computer Graphics Formats



JPEG - a photo of a cat with the compression rate decreasing, and hence quality increasing, from left to right [1].

# 2-D Computer Graphics Formats

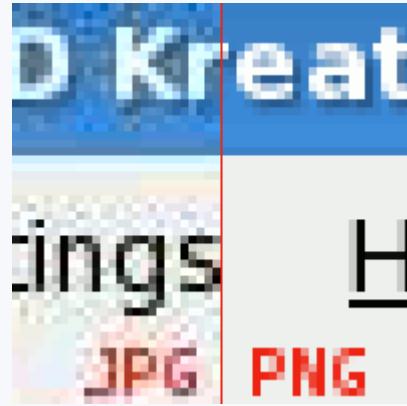


Photoshop – Save for Web menu

# Formats

- **PNG** (Portable Network Graphics) - \*.png
  - the most widely used lossless image compression format on the Internet [1]
  - created as an improved, non-patented replacement for Graphics Interchange Format (GIF) [1]
  - offers 16.8 million colors and transparency, but you can choose to use fewer colors to save file space (PNG 8, or PNG with 8-bit color) [2]
  - Lossless compression format [2]
  - common for a wide range of images, including *favicons* (the small web page icons in browser tabs) [2]
  - PNG files can be very small, but for photographs with many colors, they may be larger than comparable JPEGs [2]
  - This format supports interlacing.[2]

# 2-D Computer Graphics Formats



Composite image comparing **lossy compression** in JPEG with **lossless compression** in PNG: the JPEG artifacts are easily visible in the background, where the PNG image has solid color [1].

# Formats

- Other formats:
  - ICO - Icon Resource File;
  - DIB - Device Independent Bitmap;
  - PCX - PC PaintBrush File Format;
  - TIFF - Tag Image File Format.

## Raster Graphics

# Huge images

- Dubai: <http://gigapan.com/gigapans/48492>
- Size: 44.88 Gigapixels



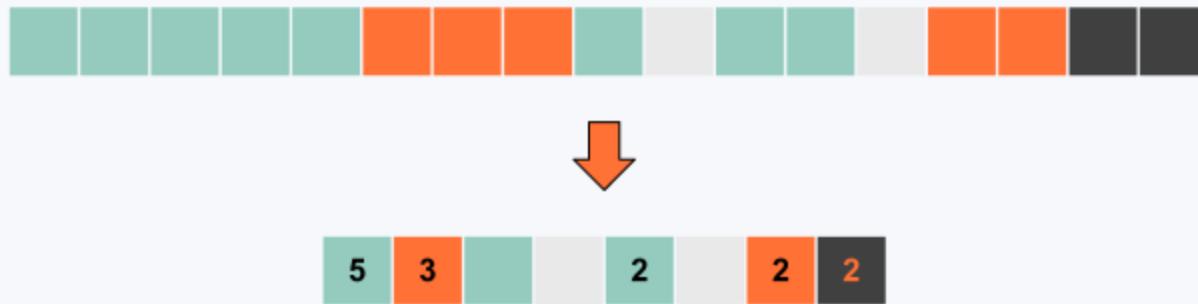
# Compression

- Compression algorithms:
  - Run-length encoding (RLE)
  - Lempel–Ziv–Welch (LZW)
  - Huffman
  - JPEG

# Compression - Run Length Encoding - RLE

- one of the simpler strategies to achieve **lossless** compression
- can be used to compress bitmapped image files (ex: \*pcx format). Bitmapped images can easily become very large because each pixel is represented with a series of bits that provide information about its color.
- RLE generates a code to “flag” the beginning of a line of pixels of the same color. That color information is then recorded just once for each pixel. In effect, RLE tells the computer to repeat a color for a given number of adjacent pixels rather than repeating the same information for each pixel over and over. The RLE compressed file will be smaller, but it will retain all the original image data—it is “lossless.”

# Compression - Run Length Encoding - RLE



# Compression - Lempel–Ziv–Welch (LZW)

- lossless data compression algorithm
- used in the GIF image format
- **Encoding**
  - Initialize the dictionary to contain all strings of length one.
  - Find the longest string W in the dictionary that matches the current input.
  - Emit the dictionary index for W to output and remove W from the input.
  - Add W followed by the next symbol in the input to the dictionary.
  - Go to Step 2.

# JPEG Compression

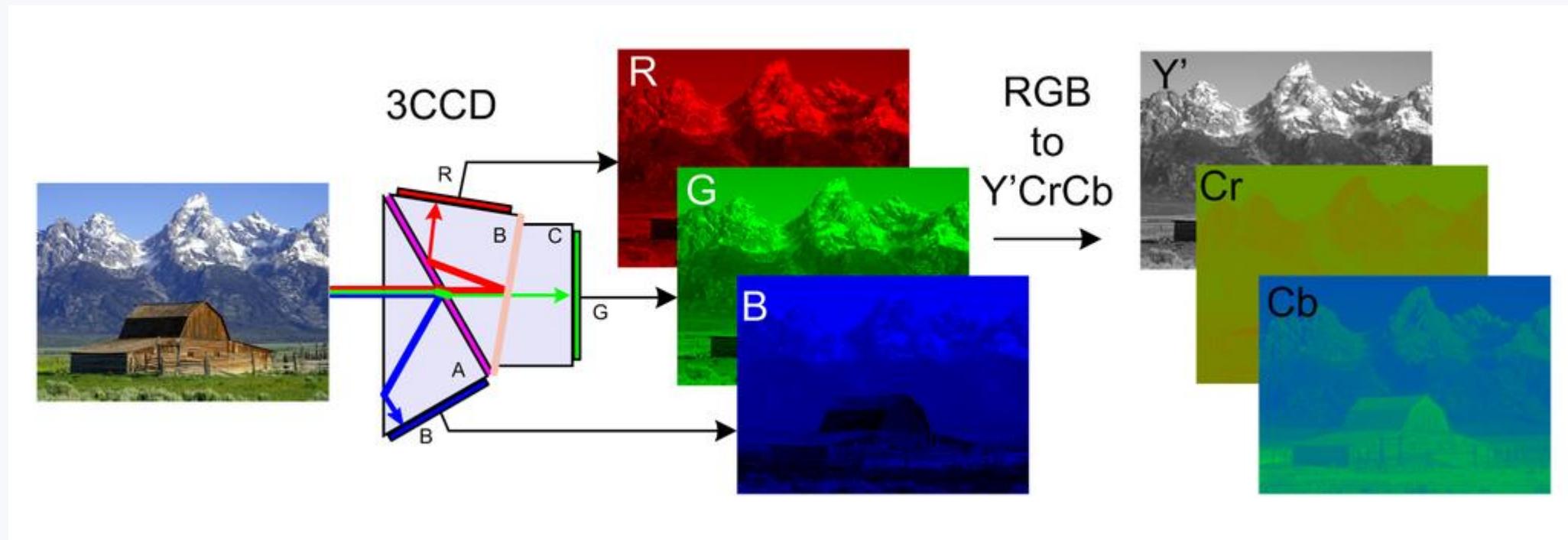
- lossy data compression algorithm

- Steps

1. color space transformation
2. downsampling
3. block splitting
4. transformation
5. quantization
6. encoding

# JPEG Compression - Step 1 - Color space transformation

- The representation of the colors in the image is converted from **RGB** to **Y'CrCb**, consisting of one luma component ( $Y'$ ), representing brightness, and two chroma components, (CB and CR), representing color. This step is sometimes skipped.



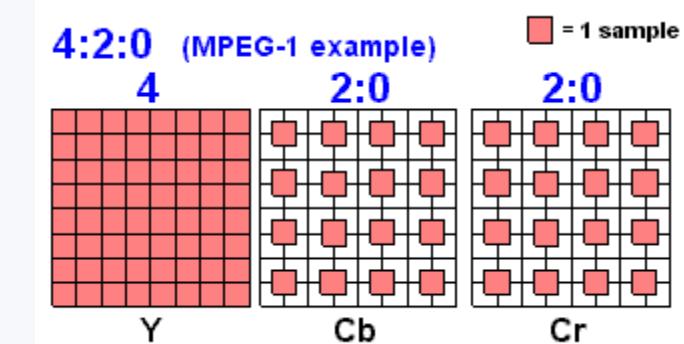
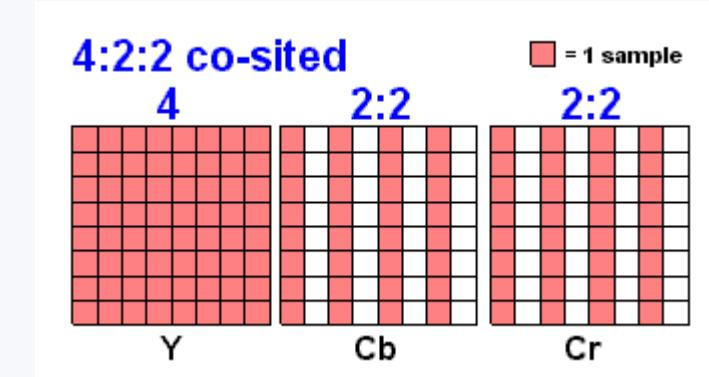
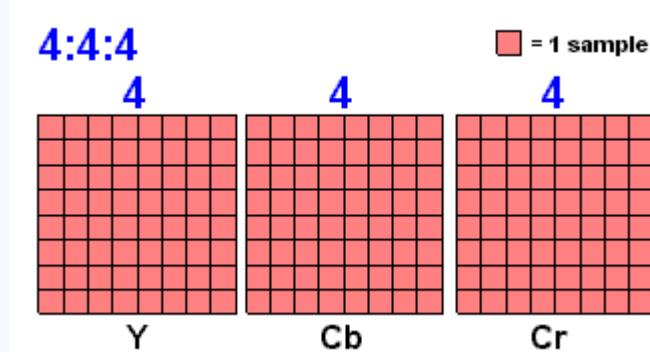
# JPEG Compression - Step 1 - Color space transformation

- The  $Y'C_BC_R$  color space conversion allows greater compression without a significant effect on perceptual image quality (or greater perceptual image quality for the same compression).
- The compression is more efficient because the brightness information, which is more important to the eventual perceptual quality of the image, is confined to a single channel. This more closely corresponds to the perception of color in the human visual system.

# JPEG Compression – Step 2 - Downsampling

- Due to the densities of color- and brightness-sensitive receptors in the human eye, humans can see considerably more fine detail in the brightness of an image (the Y' component) than in the hue and color saturation of an image (the C<sub>b</sub> and C<sub>r</sub> components). Using this knowledge, encoders can be designed to compress images more efficiently.
- The transformation into the Y'C<sub>B</sub>C<sub>R</sub> color model enables the next usual step, which is to reduce the spatial resolution of the C<sub>b</sub> and C<sub>r</sub> components.

# JPEG Compression – Step 2 - Downsampling



## JPEG Compression – Step 2 - Downsampling

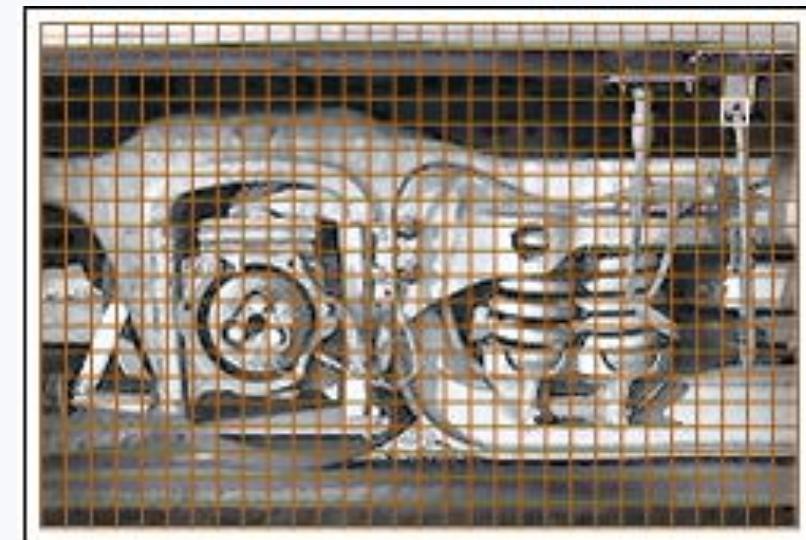
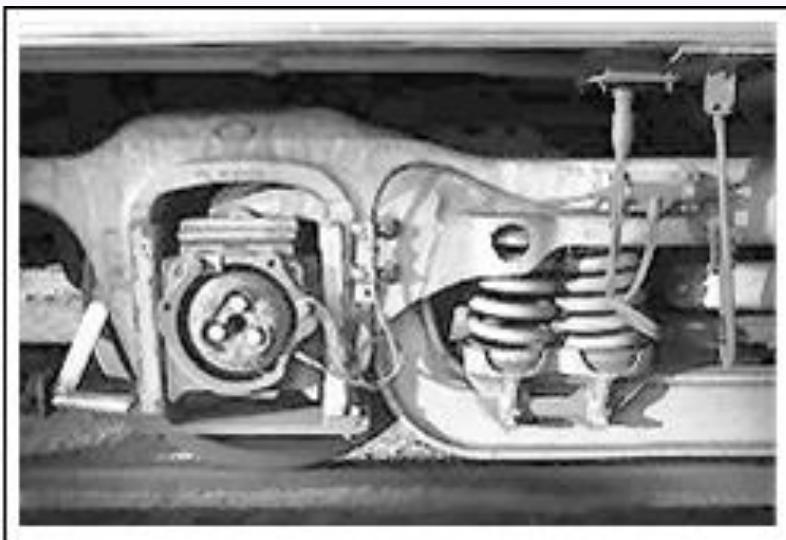
- The ratios at which the downsampling is ordinarily done for JPEG images are:
  - 4:4:4 - no downsampling,
  - 4:2:2 - reduction by a factor of 2 in the horizontal direction
  - 4:2:0 - reduction by a factor of 2 in both the horizontal and vertical directions (most common).
- For the rest of the compression process, Y', Cb and Cr are processed separately and in a very similar manner.

## JPEG Compression – Step 3 - Block splitting

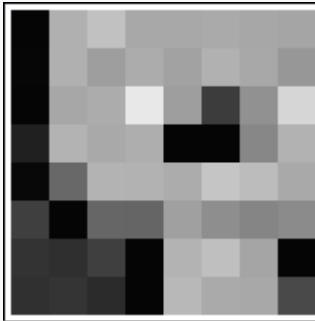
- Each channel must be split into  $8 \times 8$  blocks
- Depending on chroma subsampling, this yields Minimum Coded Unit (MCU) blocks of size  $8 \times 8$  (4:4:4 – no subsampling),  $16 \times 8$  (4:2:2), or most commonly  $16 \times 16$  (4:2:0).

# JPEG Compression – Step 4 - Discrete cosine transform

- Each channel must be split into blocks of size 8 x 8 pixels
- If we have chosen an image whose dimensions are  $160 \times 240 = 8*20 \times 8*30$ . So this step creates  $20 \times 30 = 600$  blocks.



# JPEG Compression – Step 4 - Discrete cosine transform



5	176	193	168	168	170	167	165
6	176	158	172	162	177	168	151
5	167	172	232	158	61	145	214
33	179	169	174	5	5	135	178
8	104	180	178	172	197	188	169
63	5	102	101	160	142	133	139
51	47	63	5	180	191	165	5
49	53	43	5	184	170	168	74

# JPEG Compression – Step 4 - Discrete cosine transform

- Before computing the DCT of the  $8 \times 8$  block, its values are shifted from a positive range to one centered on zero.
- We subtract 127 from each pixel intensity in each block. This step centers the intensities about the value 0 and it is done to simplify the mathematics of the transformation and quantization steps.

# JPEG Compression – Step 4 - Discrete cosine transform

5	176	193	168	168	170	167	165
6	176	158	172	162	177	168	151
5	167	172	232	158	61	145	214
33	179	169	174	5	5	135	178
8	104	180	178	172	197	188	169
63	5	102	101	160	142	133	139
51	47	63	5	180	191	165	5
49	53	43	5	184	170	168	74

-122	49	66	41	41	43	40	38
-121	49	31	45	35	50	41	24
-122	40	45	105	31	-66	18	87
-94	52	42	47	-122	-122	8	51
-119	-23	53	51	45	70	61	42
-64	-122	-25	-26	33	15	6	12
-76	-80	-64	-122	53	64	38	-122
-78	-74	-84	-122	57	43	41	-53

## JPEG Compression – Step 4 - Discrete cosine transform

- The JPEG Image Compression Standard relies on the *Discrete Cosine Transformation (DCT)* to transform the image. The DCT is applied to each  $8 \times 8$  block.
- The DCT is a product  $C = U B U^T$  where  $B$  is an  $8 \times 8$  block from the preprocessed image and  $U$  is a special  $8 \times 8$  matrix.
- Loosely speaking, the DCT tends to push most of the high intensity information (larger values) in the  $8 \times 8$  block to the upper left-hand of  $C$  with the remaining values in  $C$  taking on relatively small values.

# JPEG Compression – Step 4 - Discrete cosine transform

-27.500	-213.468	-149.608	-95.281	-103.750	-46.946	-58.717	27.226
168.229	51.611	-21.544	-239.520	-8.238	-24.495	-52.657	-96.621
-27.198	-31.236	-32.278	173.389	-51.141	-56.942	4.002	49.143
30.184	-43.070	-50.473	67.134	-14.115	11.139	71.010	18.039
19.500	8.460	33.589	-53.113	-36.750	2.918	-5.795	-18.387
-70.593	66.878	47.441	-32.614	-8.195	18.132	-22.994	6.631
12.078	-19.127	6.252	-55.157	85.586	-0.603	8.028	11.212
71.152	-38.373	-75.924	29.294	-16.451	-23.436	-4.213	15.624

## JPEG Compression – Step 5 - Quantization

- elements near zero will converted to zero and other elements will be shrunk so that their values are closer to zero. All quantized values will then be rounded to integers.
- quantization is performed in order to obtain integer values and to convert a large number of the values to 0. The Huffman coding algorithm will be much more effective with quantized data and the hope is that when we view the compressed image, we haven't given up too much resolution.

# JPEG Compression – Step 5 - Quantization

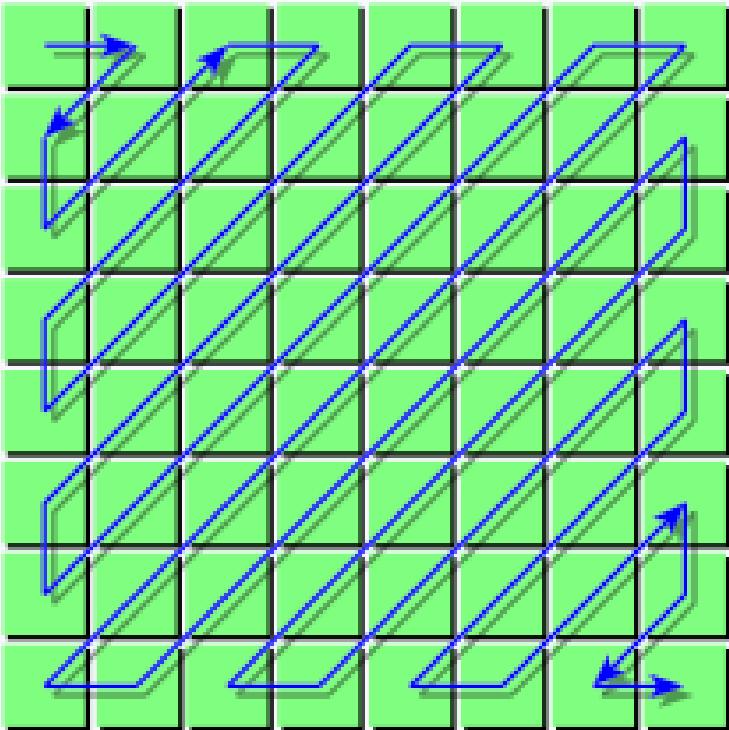
16	11	10	16	24	40	51	61
12	12	14	19	26	58	60	55
14	13	16	24	40	57	69	56
14	17	22	29	51	87	80	62
18	22	37	56	68	109	103	77
24	35	55	64	81	104	113	92
49	64	78	87	103	121	120	101
72	92	95	98	112	100	103	99

Note that the values are largest in the lower right corner of Z. These divides should produce values close to zero so that the rounding function will convert the quotient to zero

# JPEG Compression – Step 5 - Quantization

-2	-19	-15	-6	-4	-1	-1	0
14	4	-2	-13	0	0	-1	-2
-2	-2	-2	7	-1	-1	0	1
2	-3	-2	2	0	0	1	0
1	0	1	-1	-1	0	0	0
-3	2	1	-1	0	0	0	0
0	0	0	-1	1	0	0	0
1	0	-1	0	0	0	0	0

# JPEG Compression – Step 6 - Encoding



It involves arranging the image components in a "zigzag" order employing run-length encoding (RLE) algorithm that groups similar frequencies together, inserting length coding zeros, and then using **Huffman** coding on what is left.

# JPEG Compression – Step 5 - Quantization

- quantization makes the JPEG algorithm an example of *lossy compression*:
  - the DCT step is completely invertible - that is, we applied the DCT to each block  $B$  by computing  $C = U B U^T$ . It turns out we can recover  $B$  by the computation  $B = U^T C U$ .
  - converting small values to 0 and rounding all quantized values are not reversible steps. The ability to recover the original image is lost.

# HTML5 Images

- represents an image in the document.
- declaring an image element:

```

```

- API: <https://developer.mozilla.org/en-US/docs/Web/API/HTMLImageElement>
- DOM interface – `HTMLImageElement`:

# HTML5 Images

- Properties:
  - width, height: the rendered width / height of the image in CSS pixels;
  - naturalWidth, naturalHeight: unsigned long representing the intrinsic width / height of the image in CSS pixels;
  - src: DOMString that reflects the src HTML attribute, containing the full URL of the image including base URI.
- Events:
  - load: fired when a resource and its dependent resources have finished loading.

# HTML5 Images

```
var image = $("<img>")
    .load(function () { $("body").append($(this)); })
    .attr("src", "media/Penguins.jpg");
```

# HTML5 Canvas Element

- Can be used to draw graphs, make photo compositions or even perform animations.
- Declaring a canvas element:

```
<canvas id= "test" style="width:250px; height:150px">  
An alternative text describing what your canvas displays.  
</canvas>
```

- API: <https://developer.mozilla.org/en-US/docs/Web/HTML/Element/canvas>
- DOM interface: HTMLCanvasElement

# HTML5 Canvas Element - Methods

- `getContext(contextType, contextAttributes);` - returns a drawing context on the canvas, or null if the context identifier is not supported;

```
var canvas = document.getElementById('test'); // sau var canvas = $('#test')[0];
var w = canvas.width, h = canvas.height;
var ctx = canvas.getContext('2d');
```

- `contextType`: DOMString containing the context identifier defining the drawing context associated to the canvas.

# HTML5 Canvas Element - Methods

- possible values for contextType:
  - "2d", leading to the creation of a `CanvasRenderingContext2D` object representing a two-dimensional rendering context.
  - "webgl" which will create a `WebGLRenderingContext` object representing a three-dimensional rendering context. This context is only available on browsers that implement WebGL version 1 (OpenGL ES 2.0).
  - "bitmaprenderer" which will create a `ImageBitmapRenderingContext` which only provides functionality to replace the content of the canvas with a given `ImageBitmap`.

# HTML5 Canvas Element - CanvasRenderingContext2D Interface

- used for drawing rectangles, text, images and other objects onto the canvas element. It provides the 2D rendering context for the drawing surface of a <canvas> element.
- API: [developer.mozilla.org/en-US/docs/Web/API/CanvasRenderingContext2D](https://developer.mozilla.org/en-US/docs/Web/API/CanvasRenderingContext2D)
- Drawing rectangles:
  - clearRect() - sets all pixels in the rectangle defined by starting point (x, y) and size (width, height) to transparent black, erasing any previously drawn content.
  - fillRect() - draws a filled rectangle at (x, y) position whose size is determined by width and height.
  - strokeRect() - paints a rectangle which has a starting point at (x, y) and has a width and an h height onto the canvas, using the current stroke style.

# HTML5 Canvas Element - CanvasRenderingContext2D Interface

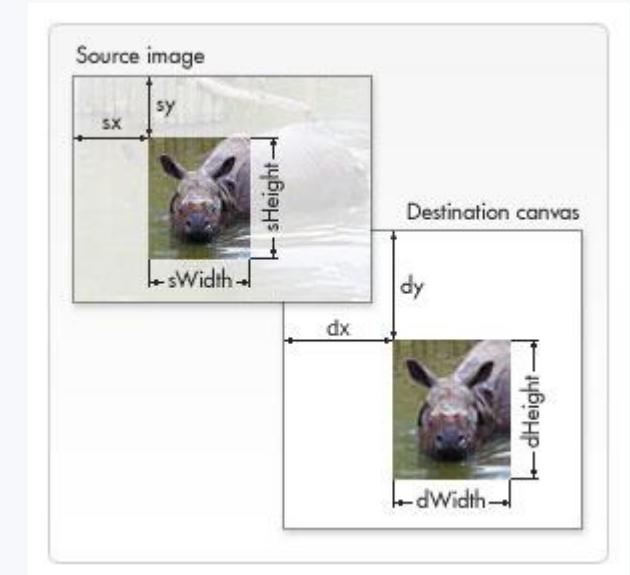
- Drawing text:
  - `fillText()` - draws (fills) a given text at the given (x,y) position;
  - `strokeText()` - draws (strokes) a given text at the given (x, y) position;
  - `measureText()` - returns a `TextMetrics` object.
- Drawing paths:
  - check [documentation](#)
- Line styles:
  - check [documentation](#)

# HTML5 Canvas Element - CanvasRenderingContext2D Interface

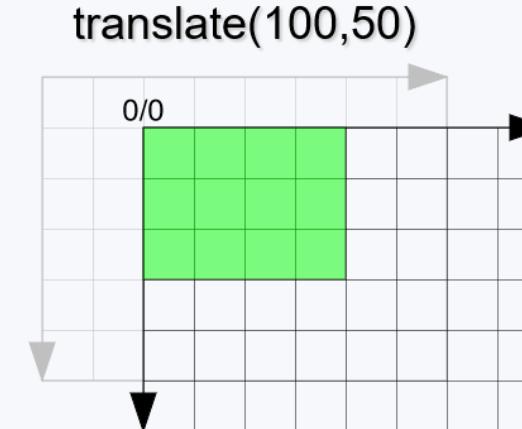
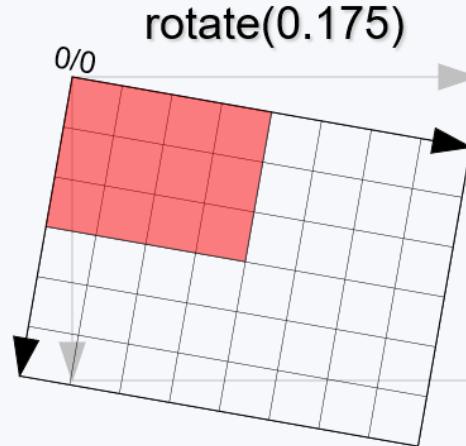
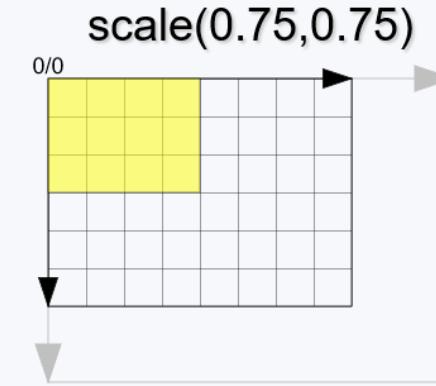
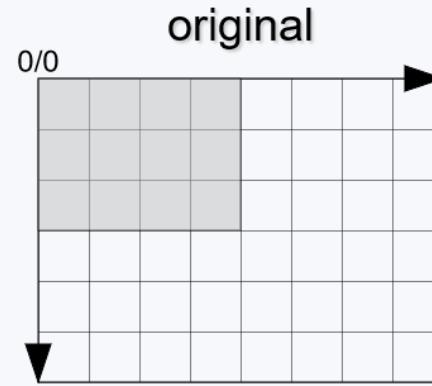
- Text styles:
  - check [documentation](#)
- Fill and stroke styles
  - check [documentation](#)
- Gradients and patterns
  - check [documentation](#)
- Shadows
  - check [documentation](#)

# HTML5 Canvas Element - CanvasRenderingContext2D Interface

- Drawing images:
  - Without scaling
    - `void ctx.drawImage(image, dx, dy);`
  - With scaling
    - `void ctx.drawImage(image, dx, dy, dWidth, dHeight);`
    - `void ctx.drawImage(image, sx, sy, sWidth, sHeight, dx, dy, dWidth, dHeight);`
  - API: [Mozilla Developer Network](#)

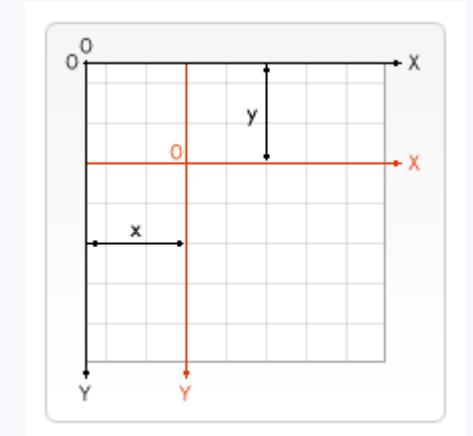


# HTML5 Canvas Element - Transformations



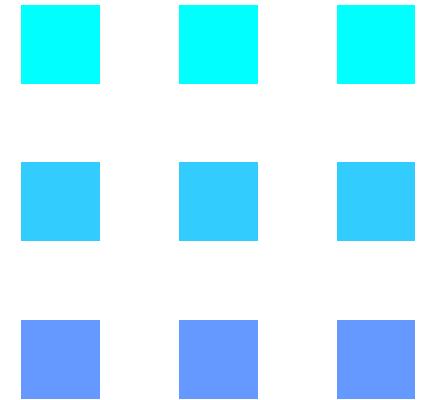
# HTML5 Canvas Element - CanvasRenderingContext2D Interface

- Translating
  - `translate(x, y)`
    - changes the origin.
    - x indicates the horizontal distance to move, and y indicates how far to move the grid vertically.
  - API: [Mozilla Developer Network](#)



# HTML5 Canvas Element - CanvasRenderingContext2D Interface

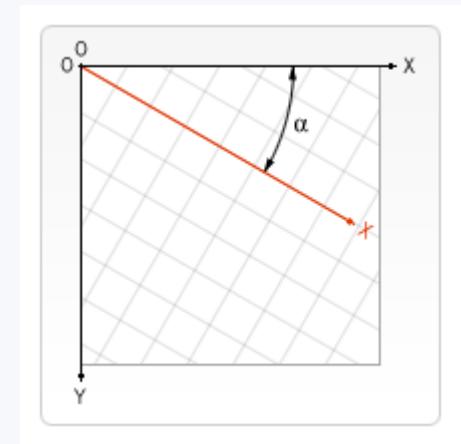
```
function draw() {  
    var ctx = document.getElementById('canvas').getContext('2d');  
    for (var i=0;i<3;i++) {  
        for (var j=0;j<3;j++) {  
            ctx.save();  
            ctx.fillStyle = 'rgb('+(51*i)+','+(255-51*i)+',255)';  
            ctx.translate(10+j*50,10+i*50);  
            ctx.fillRect(0,0,25,25);  
            ctx.restore();  
        }  
    }  
}
```



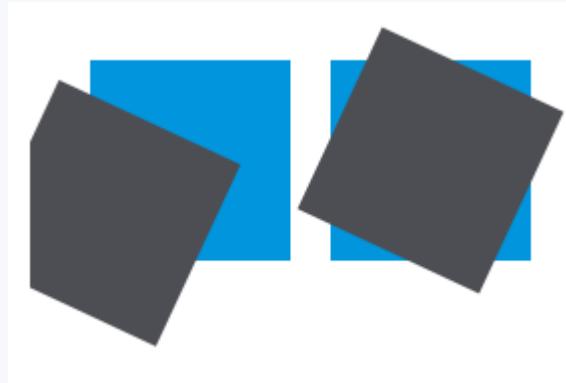
- Without the translate() method, all of the rectangles would be drawn at the same position (0,0). Using the translate() method we don't have to manually adjust the coordinates in the fillRect() function.
- JSFiddle: <https://jsfiddle.net/liviucotfas/9tfdegn7/>

# HTML5 Canvas Element - CanvasRenderingContext2D Interface

- Rotating
  - `rotate(angle)`
    - Rotates the canvas clockwise around the current origin by the angle number of radians.
    - The rotation center point is always the canvas origin, unless it has been changed using the `translate()` method
    - Note: Angles are in radians, not degrees. Convert using:  $\text{radians} = (\text{Math.PI}/180) * \text{degrees}$ .
    - API: [Mozilla Developer Network](#)



# HTML5 Canvas Element - CanvasRenderingContext2D Interface

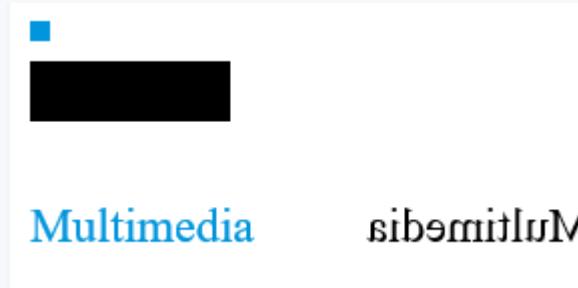


- rectangle 1: rotated based on the canvas origin
- rectangle 2: rotated from the center of the rectangle itself with the help of `translate()` method.
- JSFiddle: <https://jsfiddle.net/liviucotfas/2yf241q1/>

# HTML5 Canvas Element - CanvasRenderingContext2D Interface

- Scaling
  - `scale(x, y)`
    - Scales the canvas units by x horizontally and by y vertically.
    - Both parameters are real numbers. Values that are smaller than 1.0 reduce the unit size and values above 1.0 increase the unit size. Values of 1.0 leave the units the same size.
    - Using negative numbers you can do axis mirroring
    - API: [Mozilla Developer Network](#)

# HTML5 Canvas Element - CanvasRenderingContext2D Interface



- rectangle: scaled
- text: mirrored.
- JSFiddle: <https://jsfiddle.net/liviucotfas/Lyycaq7gv/>

# HTML5 Canvas Element - CanvasRenderingContext2D Interface

- Combining Transforms
  - the browser is using a transformation matrix

```
context.scale(-.5, 1);
context.rotate(45 * Math.PI / 180);
context.translate(40, 10);
```

- the translate, rotate, and scale methods end up affecting the values stored by this matrix

$$\begin{pmatrix} m_{11} & m_{21} & d_x \\ m_{12} & m_{22} & d_y \\ 0 & 0 & 1 \end{pmatrix}$$

# HTML5 Canvas Element - CanvasRenderingContext2D Interface

- Transforms
  - allow modifications directly to the transformation matrix
  - `transform(m11, m12, m21, m22, dx, dy)`
    - multiplies the current transformation matrix with the matrix described by its arguments.
  - API: [Mozilla Developer Network](#)

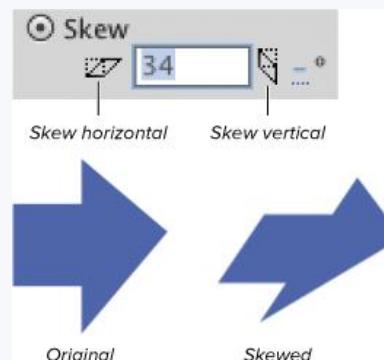
$$\begin{pmatrix} m_{11} & m_{21} & d_x \\ m_{12} & m_{22} & d_y \\ 0 & 0 & 1 \end{pmatrix}$$

# HTML5 Canvas Element - CanvasRenderingContext2D Interface

- m11 - Horizontal scaling.
- m12 - Horizontal skewing.
- m21 - Vertical skewing.
- m22 - Vertical scaling.
- dx - Horizontal moving.
- dy - Vertical moving.

transform(m11, m12, m21, m22, dx, dy)

$$\begin{pmatrix} m_{11} & m_{21} & d_x \\ m_{12} & m_{22} & d_y \\ 0 & 0 & 1 \end{pmatrix}$$



# HTML5 Canvas Element - CanvasRenderingContext2D Interface

- ex: coordinates transformation

$$\begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = \begin{pmatrix} m_{11} & m_{21} & d_x \\ m_{12} & m_{22} & d_y \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$$

- resetTransform()
  - resets the current transform to the identity matrix. This is the same as calling:  
`ctx.setTransform(1, 0, 0, 1, 0, 0);`

# HTML5 Canvas Element - CanvasRenderingContext2D Interface

- `transform(m11, m12, m21, m22, dx, dy)`
  - resets the current transform to the identity matrix, and then invokes the `transform()` method with the same arguments.
  - basically undoes the current transformation, then sets the specified transform, all in one step.

# HTML5 Canvas Element - ImageData Interface

- The ImageData interface represents the underlying pixel data of an area of a <canvas> element.
- API: [Mozilla Developer Network](#)

# HTML5 Canvas Element - ImageData Interface

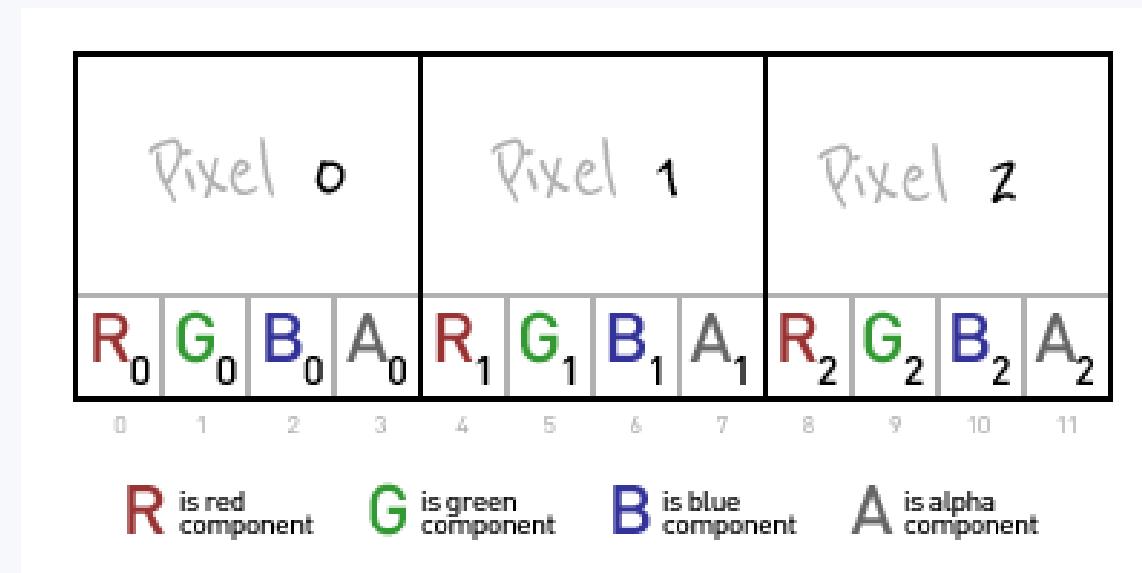
- It is created using the creator methods on the CanvasRenderingContext2D object associated with a canvas:
  - `CanvasRenderingContext2D.createImageData()`: creates a new, blank ImageData object with the specified dimensions. All of the pixels in the new object are transparent black.
  - `CanvasRenderingContext2D.getImageData(sx, sy, sw, sh)`: returns an ImageData object representing the underlying pixel data for the area of the canvas denoted by the rectangle which starts at (sx, sy) and has an sw width and sh height

# HTML5 Canvas Element - ImageData Interface

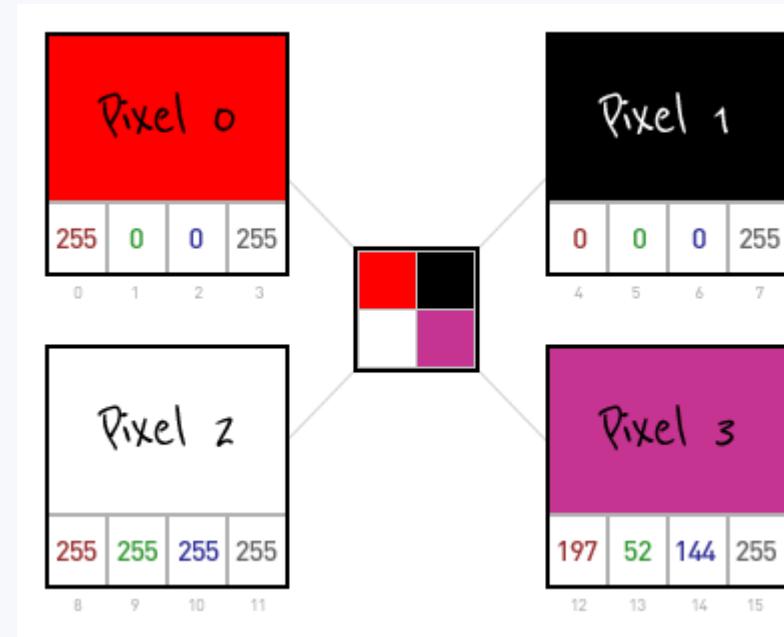
- It can also be used to set a part of the canvas by using:
  - `void ctx.putImageData(imagedata, dx, dy)`: paints data from the given `ImageData` object onto the bitmap at the given coordinates.
- JSFiddle: <https://jsfiddle.net/liviucotfas/64fzcea/>

# HTML5 Canvas Element - ImageData Interface

- Properties
  - `ImageData.data` - a `Uint8ClampedArray` representing a one-dimensional array containing the data in the RGBA order, with integer values between 0 and 255 (included).



# HTML5 Canvas Element - ImageData Interface



```
[255,0,0,255, 0,0,0,255, 255,255,255,255, 203,53,148,255]
```

# HTML5 Canvas Element - ImageData Interface

- `ImageData.height` - an unsigned long representing the actual height, in pixels, of the `ImageData`.
- `ImageData.width` - an unsigned long representing the actual width, in pixels, of the `ImageData`.

# HTML5 Canvas Element - ImageData Interface

- iterating over all the pixels in a canvas

```
var imageData = context.getImageData(0, 0, canvas.width, canvas.height);

for (var y = 0; y < canvas.height; y++) {
    for (var x = 0; x < canvas.width; x++) {
        var i = (y * canvas.width * 4) + x * 4;

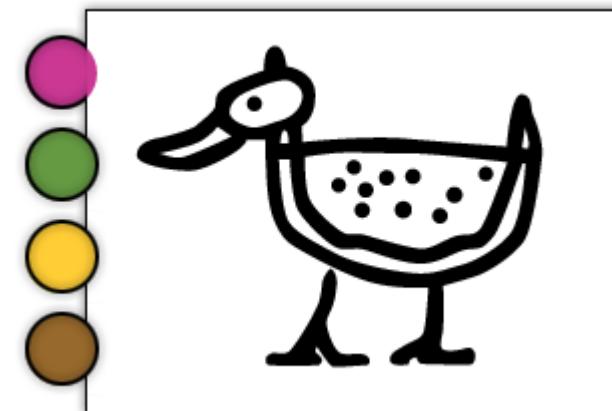
        var red = imageData.data[i]; // [0..255]
        var green = imageData.data[i+1]; // [0..255]
        var blue = imageData.data[i+2]; // [0..255]
        var transparency = imageData.data[i+3]; // [0..255]
    }
}
```

# HTML5 Canvas Element

- Examples:
  - Color picker: [https://developer.mozilla.org/en-US/docs/Web/API/Canvas\\_API/Tutorial/Pixel manipulation with canvas#A color picker](https://developer.mozilla.org/en-US/docs/Web/API/Canvas_API/Tutorial/Pixel manipulation with canvas#A color picker)
  - Zoom: [https://developer.mozilla.org/en-US/docs/Web/API/Canvas\\_API/Tutorial/Pixel manipulation with canvas#A color picker](https://developer.mozilla.org/en-US/docs/Web/API/Canvas_API/Tutorial/Pixel manipulation with canvas#A color picker)

# HTML5 Canvas Element

- Examples:
  - <http://www.williammalone.com/articles/html5-canvas-javascript-paint-bucket-tool/>
  - [www.williammalone.com/articles/create-html5-canvas-javascript-drawing-app/#demo-complete](http://www.williammalone.com/articles/create-html5-canvas-javascript-drawing-app/#demo-complete)



# Effects

- Color Filter

- Red filter:  $r' = r; g' = 0; b' = 0$

- Negative

- $r' = 255 - r; g' = 255 - g; b' = 255 - b;$

- Greyscale

- $r' = g' = b' = (r + g + b) / 3$

- $r' = g' = b' = 0.299 * r + 0.587 * g + 0.114 * b$

# Effects

- Brightness
  - $r' = r + \text{value};$  if ( $r' > 255$ )  $r' = 255$  else if ( $r' < 0$ )  $r' = 0;$
  - $g' = g + \text{value};$  if ( $g' > 255$ )  $g' = 255$  else if ( $g' < 0$ )  $g' = 0;$
  - $b' = b + \text{value};$  if ( $b' > 255$ )  $b' = 255$  else if ( $b' < 0$ )  $b' = 0;$
- Threshold
  - $v = (0.2126*r + 0.7152*g + 0.0722*b) \geq \text{threshold} ? 255 : 0; r' = g' = b' = v$

# Effects

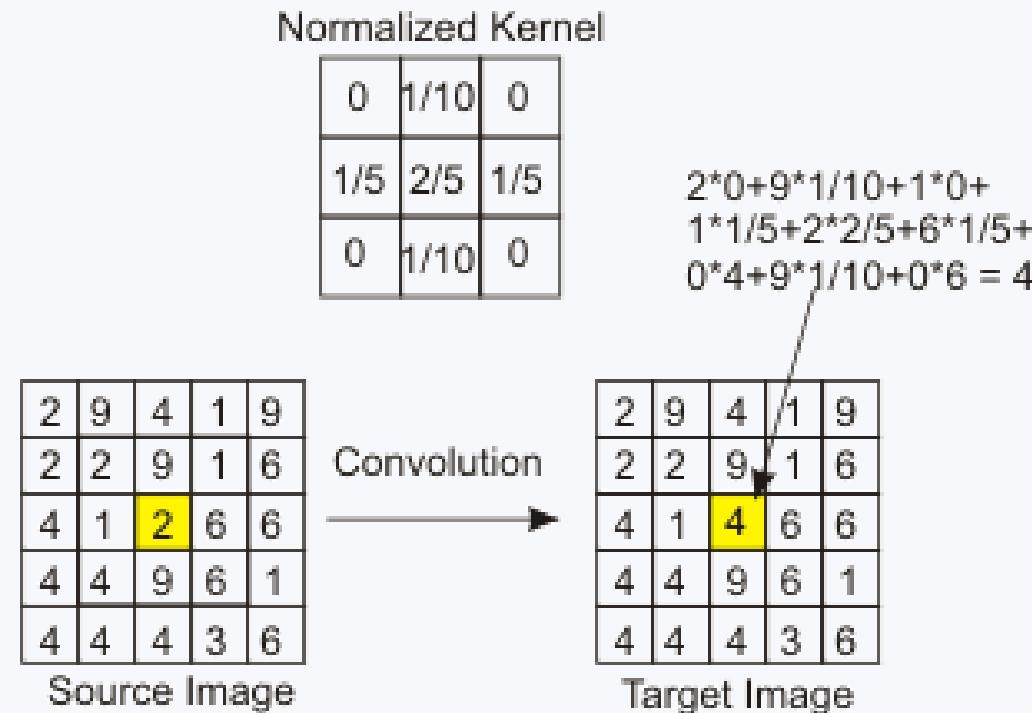
- Examples:
  - <https://www.html5rocks.com/en/tutorials/canvas/imagefilters/#toc-simplefilters>
- Further reading / examples:
  - [https://developer.mozilla.org/en-US/docs/Web/API/Canvas\\_API/Tutorial/Pixel manipulation with canvas](https://developer.mozilla.org/en-US/docs/Web/API/Canvas_API/Tutorial/Pixel manipulation with canvas)

# Effects - Convolution filters

- Allow us to implement generic filters for image processing: blurring, sharpening, embossing, edge detection.
- Calculates the new value for a pixel based on the values of nearby pixels in the original image.
- Example: blur filter 
$$\begin{bmatrix} 1/9, 1/9, 1/9, \\ 1/9, 1/9, 1/9, \\ 1/9, 1/9, 1/9 \end{bmatrix}$$
- Note: to maintain the brightness of the image, the sum of the matrix values should be one.

# Effects - Convolution filters

- Examples: <https://www.html5rocks.com/en/tutorials/canvas/imagefilters/#toc-convolution>



# WebGL

- Documentation: [https://developer.mozilla.org/en-US/docs/Web/API/WebGL\\_API](https://developer.mozilla.org/en-US/docs/Web/API/WebGL_API)
  
- Three.js
  - cross-browser JavaScript library and Application Programming Interface (API) used to create and display animated 3D computer graphics in a web browser
  - Documentation: <https://threejs.org/>
  - Example: <https://heraclosgame.com/>

## Raster Graphics

### WebGL

- Babylonjs
  - JavaScript framework for building 3D games and experiences with HTML5, WebGL, WebVR and Web Audio
  - Documentation: <http://babylonjs.com/>

# Vector Graphics

# Vector Graphics

- In **bitmapped graphics**, the computer is given a detailed description of an image that it then matches, pixel by pixel.
- In **vector-drawn graphics**, the computer is given a **set of commands** that it executes to draw the image. Pictures contain *paths* made up of points, lines, curves and shapes.
- A **vector** is a line with a particular length, curvature, and direction. **Vector graphics** are composed of lines that are mathematically defined to form shapes such as rectangles, circles, triangles, and other polygons.

# Vector Graphics

- Each vector path forms the outline of a geometric region containing color information.

# Vector Graphics



# Vector Graphics

- Because paths can be mathematically resized, vector graphics can be scaled up or down without losing any picture clarity.

# Scaling / Zoom

GIMP



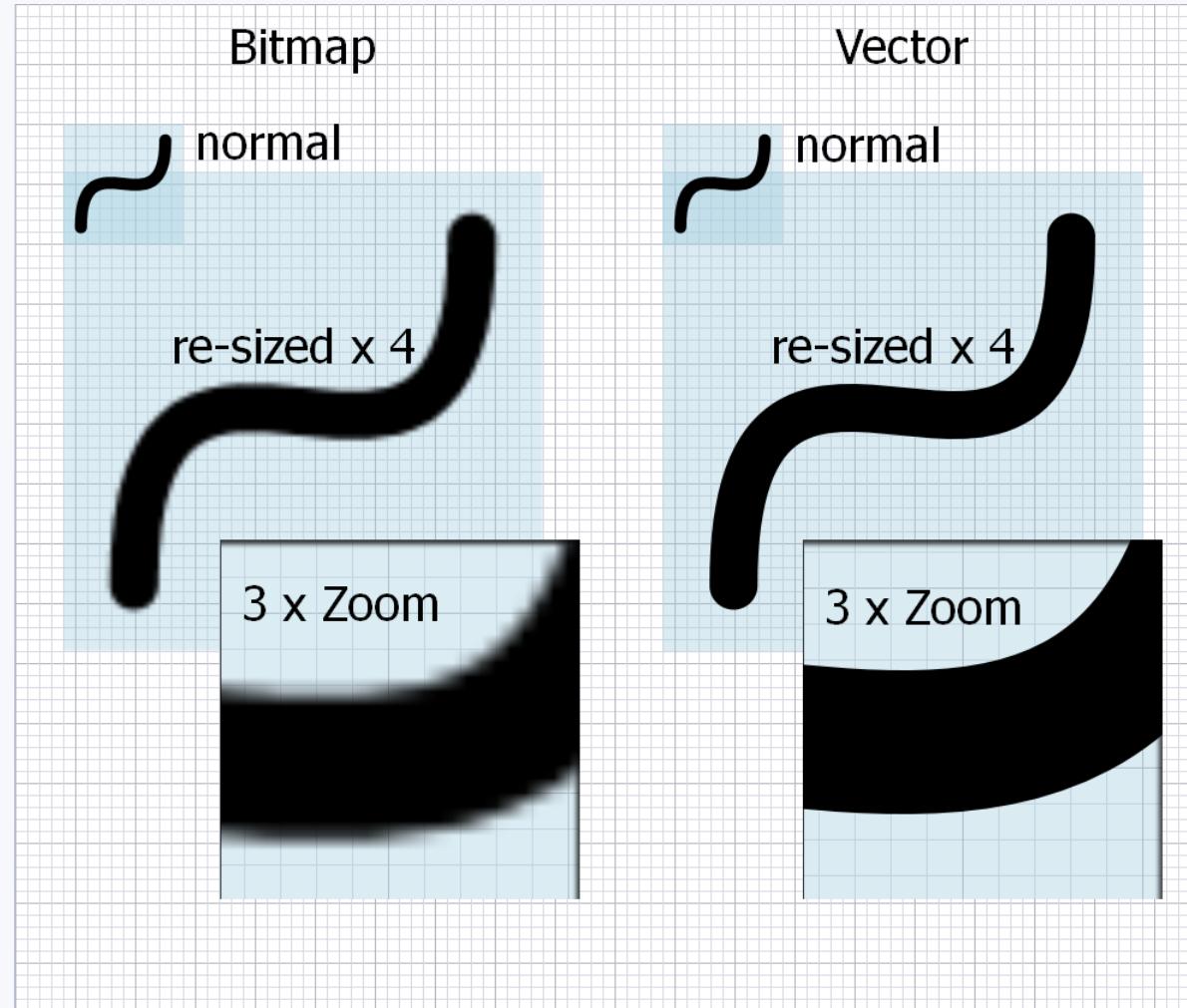
INKSCAPE



**BITMAP**  
.jpeg .gif .png

**OUTLINE**  
.svg

# Scaling / Zoom



# Animation

- You could also use vector graphics to create an animation, such as with Flash. Instead of drawing every separate frame of your project—with 24 frames appearing each second—you could create two different graphics for a segment and let your animation software mathematically interpolate the positions of the components in the in-between frames (a technique known as tweening).

# Advantages and Disadvantages

- Advantages:
  - For relatively simple images, the list of drawing commands takes up much less file space than a bitmapped version of the same graphic.
  - A draw program might use a command similar to "RECT 300, 300, RED" to create a red square with sides of 300 pixels. The file for this image contains 15 bytes that encode the alphanumeric information in the command.
  - The same image could also be created with a paint program as a bitmapped graphic. Using 8-bit color resolution (one byte per pixel), this file would require 90,000 bytes ( $300 \times 300$ ). The much smaller files sizes of drawn images can be a significant advantage for multimedia developers.

# Advantages and Disadvantages

- Another advantage of vector-drawn graphics is smooth **scaling**. Vector images are enlarged by changing the parameters of their component shapes. The new image can then be accurately redrawn at the larger size without the distortions typical of enlarged bitmapped graphics.
- Images that are needed in several sizes, such as a company logo, are often best handled as vector-drawn graphics.

# Advantages and Disadvantages

- Disadvantages:
  - The principal disadvantage of vector-drawn graphics is lack of control over the individual pixels of the image. As a result, draw programs cannot match the capabilities of bitmapped applications for display and editing of photo-realistic images.

# Formats

- SVG (Scalable Vector Graphics)
  - XML-based vector image format for two-dimensional graphics
  - provides supports for interactivity and animation
- DXF (Drawing Exchange Format)
  - is a CAD data file format developed by Autodesk for enabling data interoperability between AutoCAD and other programs.

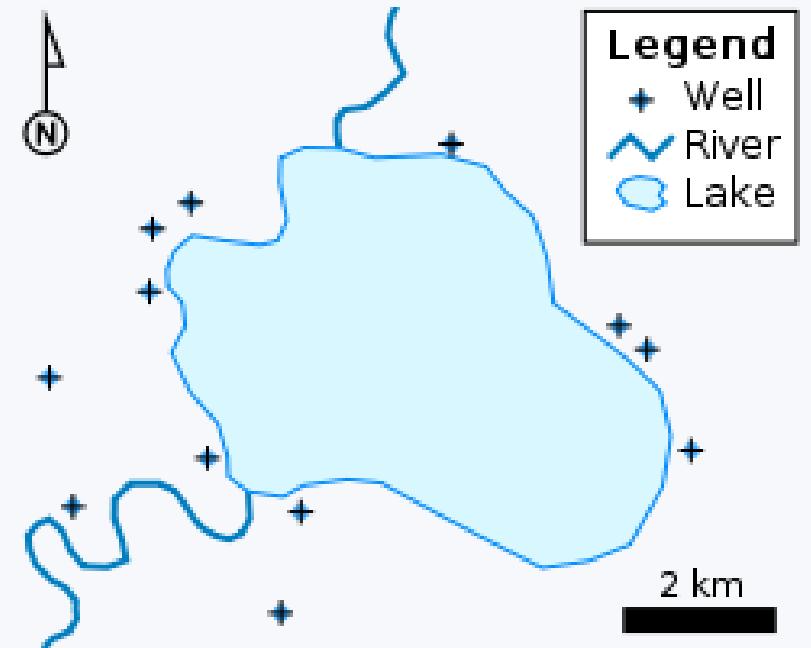
## Formats

- EPS (Encapsulated Post Script)
  - Created by Adobe Systems for representing vector graphics
  - Uses a computer language called Post Script
- SHP (Shapefile)
  - developed and regulated by Esri as a (mostly) open specification for data interoperability among Esri and other GIS software products
  - popular geospatial vector data format for geographic information system (GIS) software

# Vector Graphics Formats

Simple vector map that can be stored as Shape file:

- points (wells),
- polylines (rivers), and
- polygons (lake).

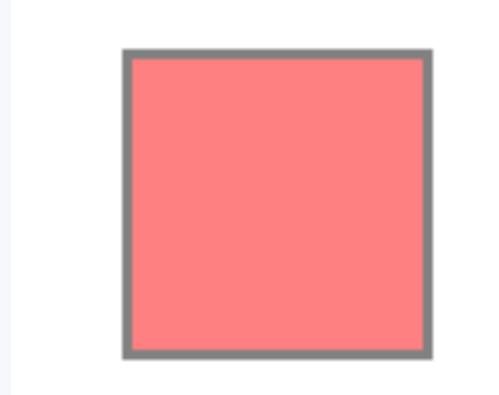


# SVG – Scalable Vector Graphics

- XML-based vector image format for two-dimensional graphics
- provides supports for interactivity and animation

```
<!DOCTYPE html>
<html>
<body>
    <svg width="400" height="180">
        <rect x="50" y="20" width="150" height="150"
style="fill:red;stroke:black;stroke-width:5;opacity:0.5">
    </svg>
</body>
```

# SVG – Scalable Vector Graphics



# SVG – Scalable Vector Graphics

- Creating SVG Images
- SVG images can be created with any text editor, but it is often more convenient to create SVG images with a drawing program, like [Inkscape](#).

# SVG – Scalable Vector Graphics

- Line

```
<line x1="start-x" y1="start-y" x2="end-x" y2="end-y">
```

- Example:

- [http://www.w3schools.com/graphics/svg\\_line.asp](http://www.w3schools.com/graphics/svg_line.asp)

```
<svg height="210" width="500">
    <line x1="0" y1="0" x2="200" y2="200" style="stroke:rgb(255,0,0);stroke-width:2" />
</svg>
```

# SVG – Scalable Vector Graphics

- Rectangle

```
<rect x="start-x" y="start-y" width="width" height="height" rx="radius-x"  
ry="radius-y"/>
```

- Example

- [http://www.w3schools.com/graphics/svg\\_rect.asp](http://www.w3schools.com/graphics/svg_rect.asp)

```
<svg width="400" height="110">  
  <rect width="300" height="100" style="fill:rgb(0,0,255);stroke-width:3;stroke:rgb(0,0,0)" />  
</svg>
```

# SVG – Scalable Vector Graphics

- Circle

```
<circle cx="center-x" cy="center-y" r="radius"/>
```

- Example
  - [http://www.w3schools.com/graphics/svg\\_circle.asp](http://www.w3schools.com/graphics/svg_circle.asp)

```
<svg height="100" width="100">
  <circle cx="50" cy="50" r="40" stroke="black" stroke-width="3" fill="red" />
</svg>
```

# SVG – Scalable Vector Graphics

- Elipse

```
<ellipse cx="center-x" cy="center-y" rx="radius-x" ry="radius-y"/>
```

- Polygon

```
<polygon points="x1,y1 x2,y2 ..."/>
```

- Polyline

```
<polyline points="x1,y1 x2,y2 ..."/>
```

- Text

```
<text x="start-x" y="start-y">continut</text>
```

# SVG – Scalable Vector Graphics

- Grouping elements

```
<g id="id_grup">... <!-- elemente --> </g>
```

- Defining groups

```
<defs>... <!-- definire grupuri --> </defs>
```

- Reusing groups

```
<use xlink:href="#id_grup" x="30" y="14"/>
```

## Vector Graphics

# SVG – Scalable Vector Graphics

- Interacting with SVG using CSS
- Link: [https://developer.mozilla.org/en-US/docs/Web/Guide/CSS/Getting started/SVG and CSS](https://developer.mozilla.org/en-US/docs/Web/Guide/CSS/Getting_started/SVG_and_CSS)

## Vector Graphics

# SVG – Scalable Vector Graphics

- Interacting with SVG using JavaScript / jQuery
  - similar to the approach used for HTML elements;
- Particularities
  - when creating an element we need to use the SVG namespace

```
document.createElementNS("http://www.w3.org/2000/svg", „TAG_SVG")
```

## Vector Graphics

# SVG – Scalable Vector Graphics

- Example

```
$(document.createElementNS("http://www.w3.org/2000/svg", "rect"))  
    .attr({x:160, y:160, width:12, height:12})  
    .appendTo($("#drawing"));
```

<https://developer.mozilla.org/en-US/docs/Web/SVG>

# Digital Video

## Definition

- Digital video is a representation of **moving visual images** in the form of encoded digital data. This is in contrast to analog video, which represents moving visual images with analog signals.
- Digital video comprises a series digital images displayed in rapid succession. In contrast, one of the key analog video methods, motion picture film, uses a series of photographs which are projected in rapid succession.
- Digital video was first introduced commercially in 1986 with the Sony D1 format, which recorded an uncompressed standard definition component video signal in digital form instead of the high-band analog forms that had been commonplace until then.

## Advantages

- Digital video can be processed using specialized computer software.
- Digital video can be **copied** with no degradation in quality. In contrast, when analog sources are copied, they experience generation loss.
- Digital video can also be **stored** on hard disks or **streamed** over **the Internet** to end users who watch content on a desktop computer screen or a digital Smart TV.
- The soundtrack is also stored as digital audio .

# Characteristics

- Screen Resolution
- Color Depth
- Frame Rate
- Interlaced / Progressive
- Compression Method (codec)

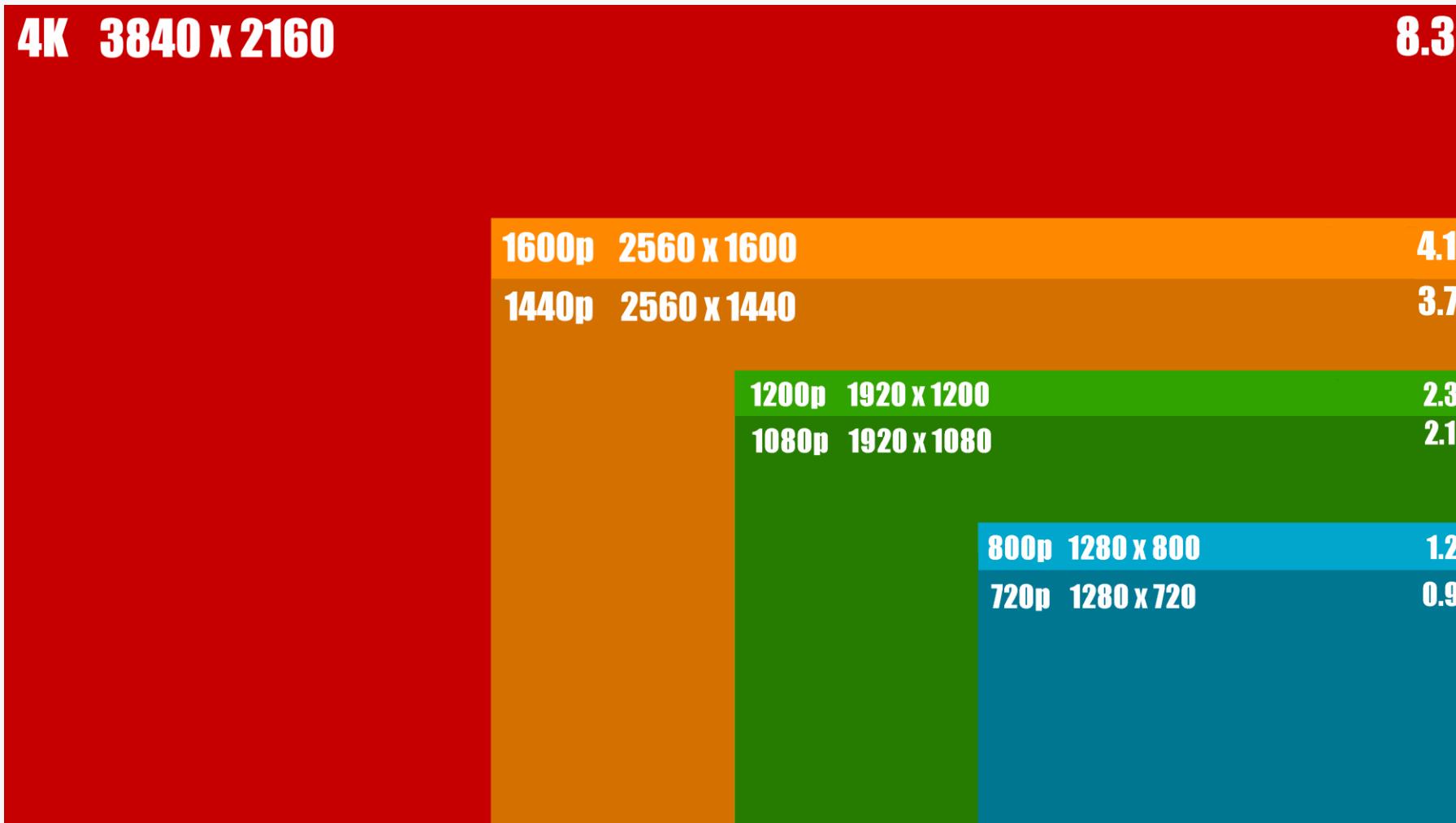
# Characteristics - Screen Resolution

- means the **width** and **height** of the displayed image, measured in pixels. In other words, the total number of pixels contained in each individual frame (**also called Spatial resolution**) [1]
- the smaller the screen resolution of a digital video, the less processing, storage, and transmission it requires [2].
- The output resolution of the relatively high-quality DV (digital video) format is **720 × 480 pixels**. This requires the computer to process and transmit information for nearly **350,000 pixels** at rates of up to 30 times per second [2].
- Reducing display size to **176 × 144** for mobile phones yields a pixel count of approximately **25,000** [2].

# Characteristics – Common Screen Resolution

Name	Pixels (width x height)	Aspect Ratio	Notes
<i>Standard Definition (SD)</i>			
<b>480p / 480i</b>	720x480 (or 704x480)	4:3 (approx)	NTSC
<b>576p / 576i</b>	720x576 (or 704x576)	4:3 (approx)	PAL
<i>High Definition (HDTV)</i>			
<b>720p</b>	1280x720	16:9	
<b>1080p / 1080i</b>	1920x1080	16:9	
<i>Ultra High Definition (UHDTV)</i>			
<b>4K (2160p)</b>	3840x2160	16:9	Exactly 4 × 1080p
<b>8K (4320p)</b>	7680x4320	16:9	Exactly 16 × 1080p
<b>8640p</b>	15360x8640	16:9	Exactly 32 × 1080p
<i>Digital Cinema (DCI)</i>			
<b>2K</b>	2048 × 1080	1.90:1	The first generation of digital cinema projectors.
<b>4K</b>	4096 × 2160	1.90:1	2nd generation digital cinema.

# Characteristics - Screen Resolution



# Characteristics - Screen Resolution



## Characteristics - Color Depth

- Color **depth** is the number of bits used to indicate the color of a single **pixel** in a video frame buffer, or the number of bits used for each color component of a single **pixel**
- Similar to raster images, the more bits are used for storing the color, the more subtle variations of colors can be reproduced.

## Characteristics – Frame Rate

- Frame rate, also known as frame frequency, is the frequency (rate) at which an imaging device displays consecutive images called frames.
- The perception of continuous motion in video is dependent on two factors. First, images must be displayed rapidly enough to allow persistence of vision to fill the interval between frames. Second, the changes in the location of objects from frame to frame must be relatively gradual.
- Lowering the frame rate both slows the delivery of individual images and drops out frames of video. If the rate is low enough, viewers will experience a string of still images with abrupt changes of content—in other words, a jerky video.

## Characteristics – Frame Rate

- A common frame rate for broadcast video is 30 frames per second (fps). Video intended for streaming over the Internet is often delivered at a rate of just 15 fps, effectively cutting the required data rate in half [1]
- NTSC used about 30 frames per second and PAL used 25 frames per second [2].

# Characteristics – Interlaced / Progressive

- **Progressive video**
  - each frame is displayed similar to the text on a page - line by line, top to bottom.
- **Interlaced video**
  - each frame is composed of two halves of an image. In the first pass all odd numbered lines are displayed, from the top left corner to the bottom right corner. The second pass displays the second and all even numbered lines, filling in the gaps in the first scan.
  - the two halves are referred to individually as fields. Two consecutive fields compose a full frame. If an interlaced video has a frame rate of 15 frames per second the field rate is 30 fields per second.

# Characteristics – Interlaced / Progressive

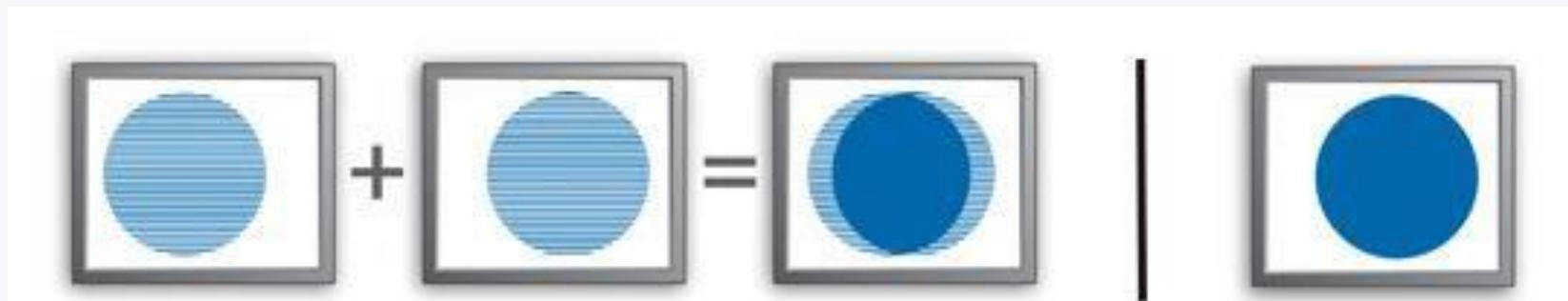


# Characteristics – Interlaced / Progressive

- Examples of formats:
  - PAL -576p (progressive) / 576i (interlaced)
  - FullHD – 1080p (progressive) / 1080i (interlaced)
- Benefits of interlacing
  - for a fixed bandwidth, interlace provides a video signal with twice the display refresh rate for a given line count (versus progressive scan video at a similar frame rate—for instance 1080i at 60 half-frames per second, vs. 1080p at 30 full frames per second).
  - bandwidth benefits **only apply** to an **analog or uncompressed digital video** signal. With digital video compression, as used in all current digital TV standards, interlacing introduces additional inefficiencies.

# Characteristics – Interlaced / Progressive

- Interlacing issues
  - Because each interlaced video frame is composed of two fields captured at different moments in time, interlaced video frames can exhibit motion artifacts known as interlacing effects, if recorded objects move fast enough to be in different positions when each individual field is captured.
  - These artifacts may be more visible when interlaced video is displayed at a slower speed than it was captured, or in still frames.



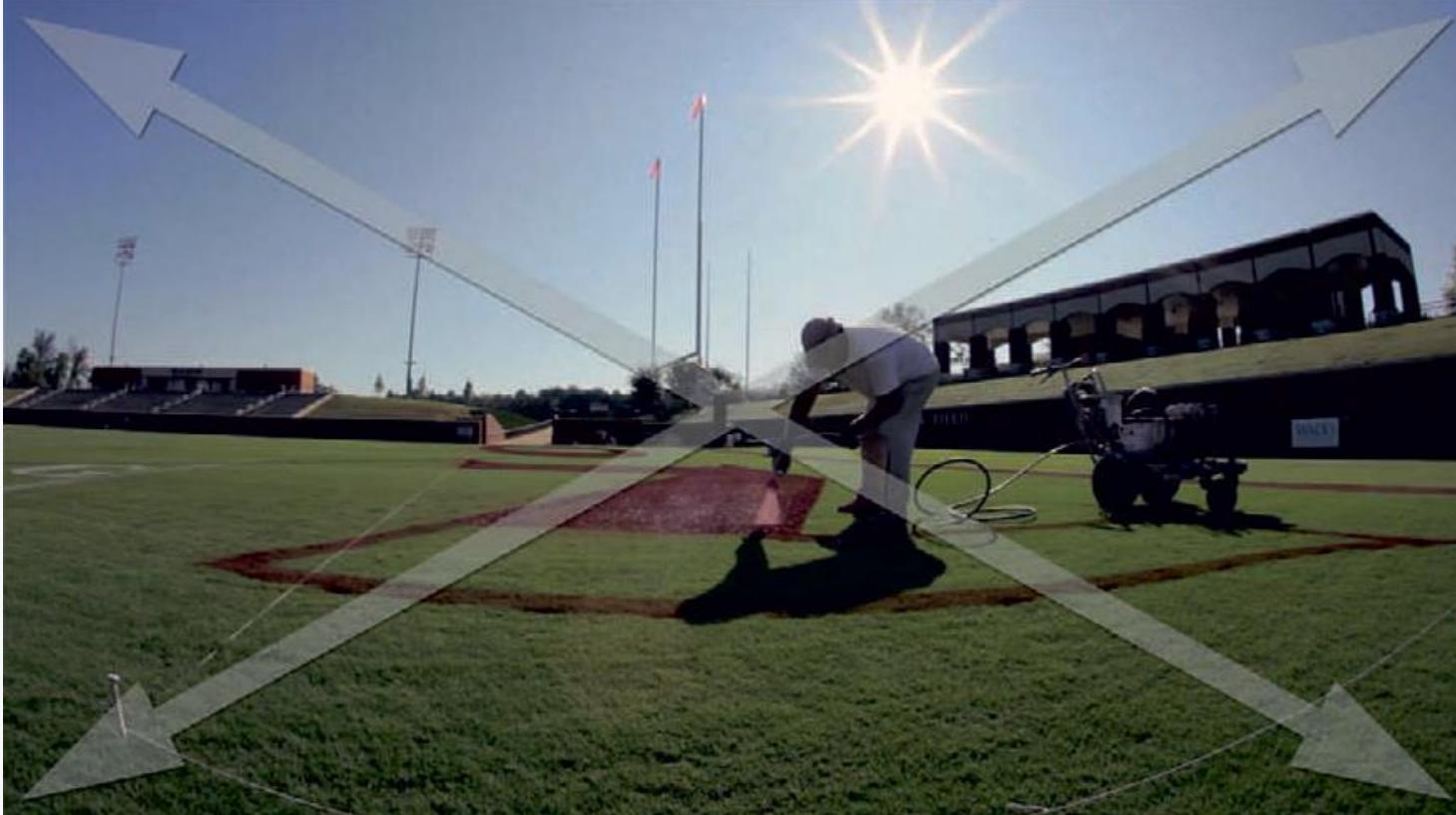
## Characteristics – Compression

- **Redundancy** - the amount of wasted space consumed by storage media to record picture information in a digital image / digital video.
- The goal of **video compression** is two-fold:
  1. to reduce the file size of an image by eliminating or rewriting as much of the redundant information as possible;
  2. to retain the visible quality of an image.

# Characteristics – Compression

- Redundancies can occur:
  - within the two-dimensional space of a single frame of video (as with a photograph) - *spatial redundancy*.
  - across time in a video sequence containing many frames – *temporal redundancy*.
- Example: a five-second (150 frames) shot of a blue sky on a cloudless day.
  - thousands of blue pixels retain the same color value across the entire sequence of 150 frames. This phenomenon is called *temporal redundancy* and occurs whenever the value of a pixel remains unchanged from one frame to the next in a time-based sequence. The pixels also stretch outward within the space of each individual frame. This phenomenon is called *spatial redundancy*.

# Characteristics – Compression



Spatial redundancy

# Characteristics – Compression



Temporal redundancy (1 second / 30 Frames)

# Characteristics – Compression

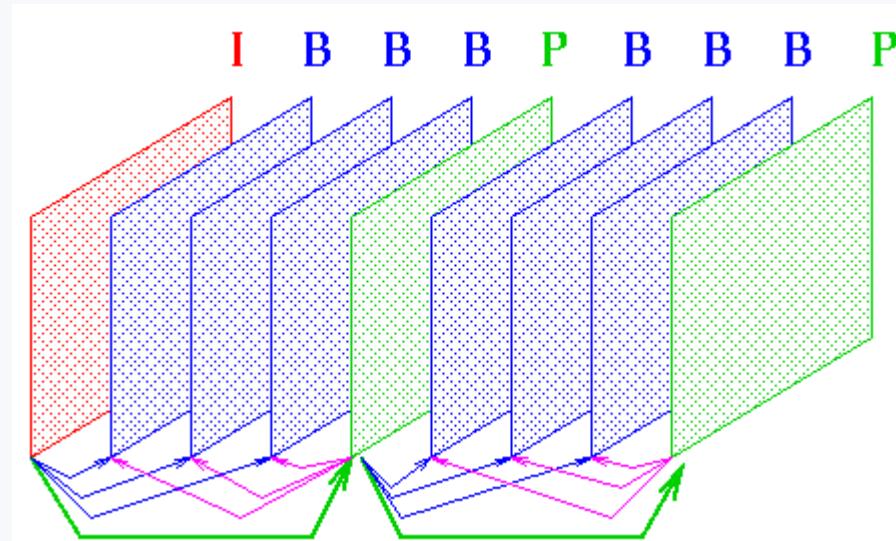
- *Intraframe (or I-frame) compression*
  - Eliminates spatial redundancies “within” a video frame in much the same way that JPEG compression is used to reduce them in a digital photograph.
  - I-frames are typically compressed at a ratio of 10:1. This means that a compressed I-frame consumes as little as 10% of the file space of a raw uncompressed frame.
  - Since I-frames are fully defined by information from within the frame, they can be easily decoded and rendered onscreen during playback and editing.
  - However, the overall amount of compression that’s achieved with this approach is limited since temporal redundancies are not addressed.

# Characteristics – Compression

- *Interframe compression* (more common method)
  - exploits both spatial and temporal redundancies.
  - using the previous method of **intraframe compression**, all of the frames in a video stream are turned into I-frames. Each one is *intracoded* to eliminate spatial redundancy. Thus, compression is applied evenly to all of the frames within a video stream.
  - with **interframe compression**, I-frames are created at fixed intervals (typically every 15 frames). An I-frame marks the beginning of a packaged sequence of adjacent frames called a GOP (Group of Pictures). The fully defined I-frame serves as a *keyframe* or reference for other frames in the group. Its job is to hold repeating color values in place that will not change across the sequence. Basically, it creates a digital marker on the frame that says “do not change the color of this pixel until you are told to do so.”

# Characteristics – Compression

- MPEG Compression
  - exploits both spatial and temporal redundancies (interframe compression )
  - each I-frame is followed by a sequence of 14 frames designated as either a *P-frame* or a *B-frame*.



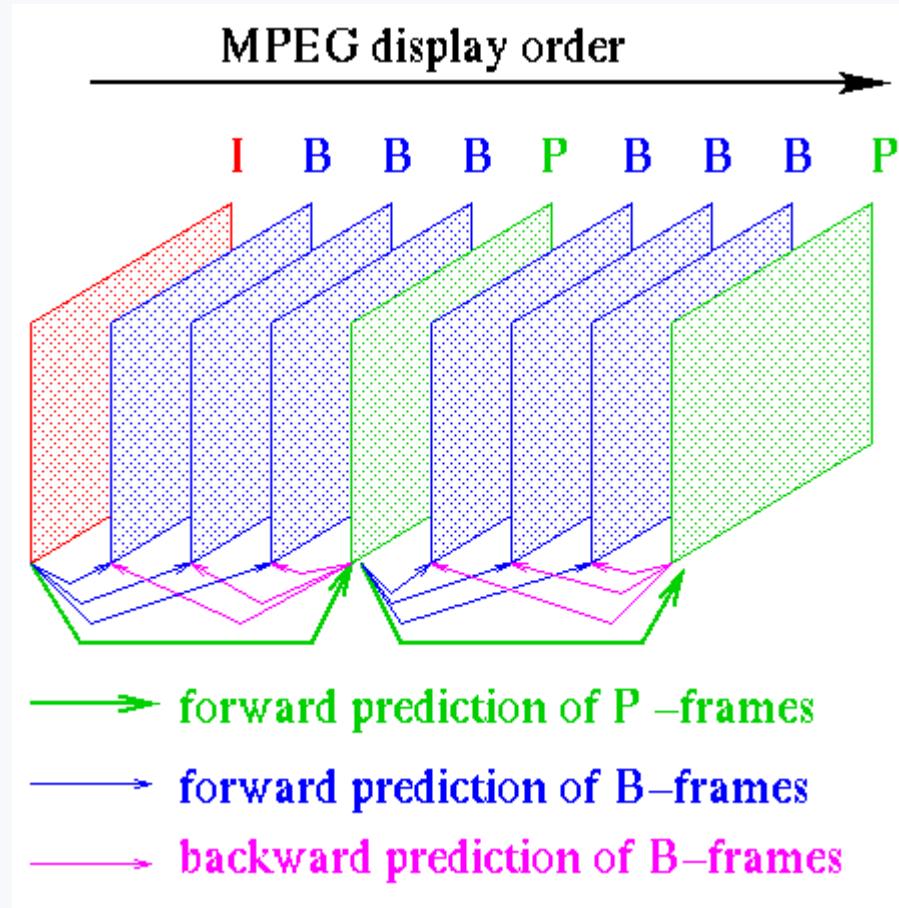
# Characteristics – Compression

- A *P-frame*
  - a predictive coded image that only stores data for pixels that are different from the preceding frame.
  - Example: in a shot of a bird flying across a blue sky, only pixels related to the moving bird would be encoded to a P-frame. The unchanged background pixels simply carry forward from information stored in the previous frame.
  - on average, a P-frame can be compressed twice as much as an I-frame.

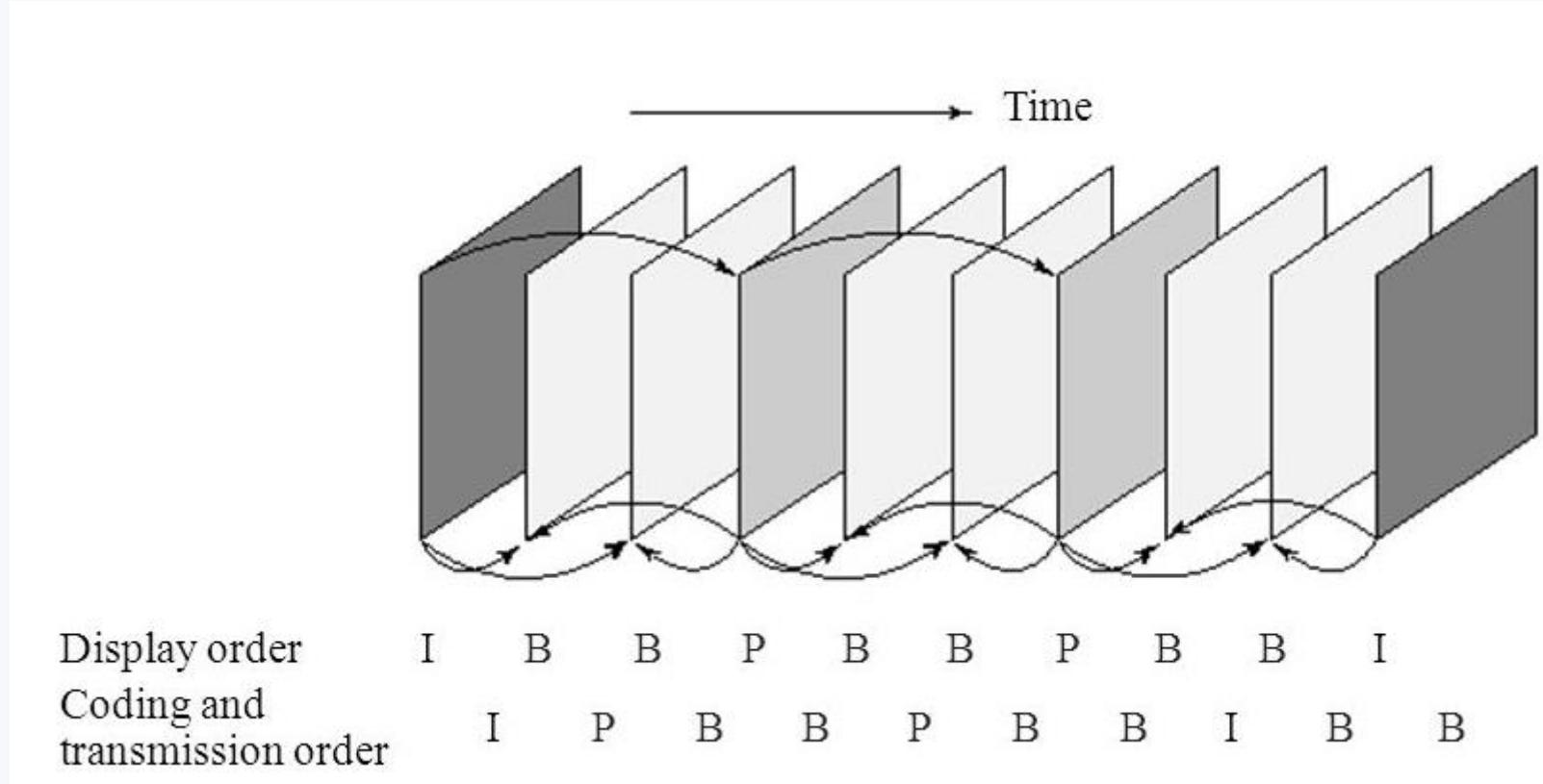
# Characteristics – Compression

- A *B-frame*
  - a bidirectional predictive coded image. It records changes from both the preceding frame and the one immediately following.
  - On average, a B-frame can be compressed twice as much as a P-frame.
- With **interframe encoding**, all frames are not equally compressed. The more heavily compressed P-frames and B-frames require more processing power to encode and decode.

# Characteristics – Compression



# Characteristics – Compression



# Characteristics – Video Formats - Containers

- Container:
  - exists solely for the purpose of bundling all of the audio, video, and codec files into one organized package.
  - in addition, the container often contains chapter information for DVD or Blu-ray movies, metadata, subtitles, and/or additional audio files such as different spoken languages.

# Characteristics – Video Formats - Containers

- **Matroska Multimedia Container MKV:**
  - file extension: .mkv .mk3d .mka .mks
  - Google \*.webm is based on the specifications of this format
  - open standard, free container format
  - supports almost any audio or video format which makes it adaptable, efficient, and highly regarded as one of the best ways to store audio and video files.
  - supports multiple audio, video and subtitle files even if they are encoded in different formats.
  - due to the options the container offers, as well as its handling of error recovery (which allows you to play back corrupted files), it has quickly become one of the best containers currently available.

## Characteristics – Video Formats - Containers

- **MPEG-4 (MP4):**
  - file extension: .mp4
  - digital multimedia container format most commonly used to store video and audio
  - recommended format for uploading video to the web
  - utilizes MPEG-4 encoding, or H.264, as well as AAC or AC3 for audio. It's widely supported on most consumer devices, and the most common container used for online video

# Characteristics – Video Formats - Containers

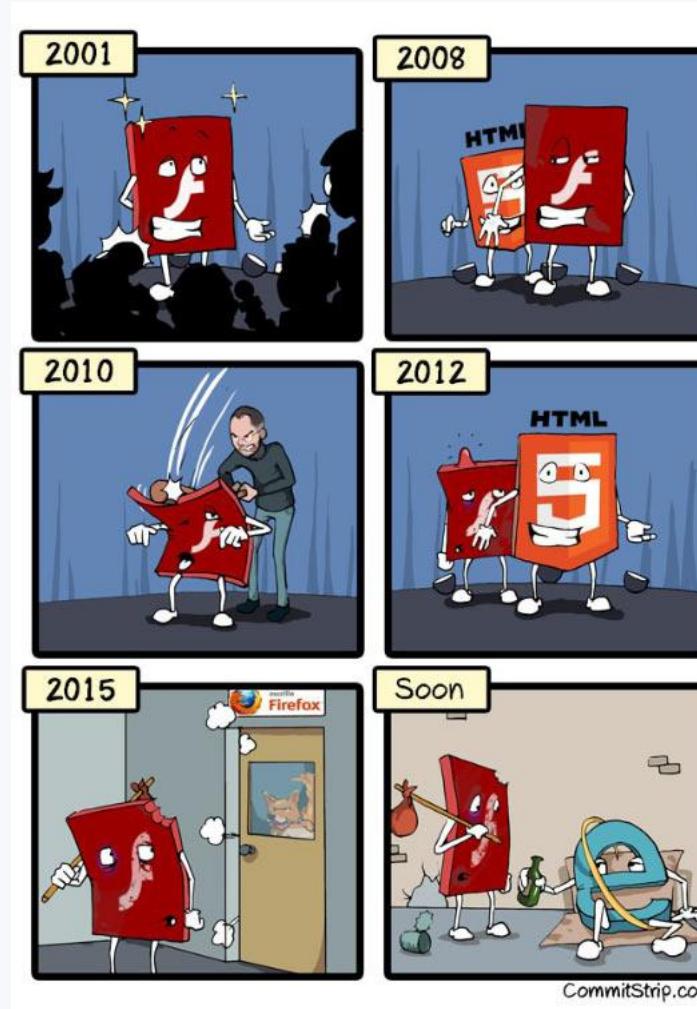
- Other popular Video Containers:
  - Ogg - free, open container format
  - AVI – Windows Professional
  - MOV – Mac everything
  - MPEG or MPG – by the MPEG group
  - FLV – Flash video
  - MP4 – by the MPEG group
  - VOB – DVDs
- Comparison:
  - [https://en.wikipedia.org/wiki/Comparison\\_of\\_video\\_container\\_formats](https://en.wikipedia.org/wiki/Comparison_of_video_container_formats)

## Web Video

*"Flash was created during the PC era – for PCs and mice. Flash is a successful business for Adobe, and we can understand why they want to push it beyond PCs. But the mobile era is about low power devices, touch interfaces and open web standards – all areas where Flash falls short. The avalanche of media outlets offering their content for Apple's mobile devices demonstrates that Flash is no longer necessary to watch video or consume any kind of web content."*

*Steve Jobs*

# Web Video - The Evolution Of Flash



# Web Video

- <video> - HTML element used to embed video content in a document. The video element contains one or more video sources. To specify a video source, use either the src attribute or the <source> element; the browser will choose the most suitable one.

```
<video controls>
  <source src="foo.webm" type="video/webm">
  <source src="foo.ogg" type="video/ogg">
  <source src="foo.mov" type="video/quicktime">
  I'm sorry; your browser doesn't support HTML5 video.
</video>
```

# Web Video

- <video>
  - attribute: *autoplay, controls, src, volume, ...*
  - API: <https://developer.mozilla.org/en-US/docs/Web/HTML/Element/video>
- HTMLVideoElement (JavaScript)
  - properties: currentSrc, currentTime, duration, ended, error, paused, readyState, volume
  - methods canPlayType, load, pause, play
  - events: canplay, ended, pause, play, volumechange, waiting
- API: <https://developer.mozilla.org/en-US/docs/Web/API/HTMLVideoElement>

# Web Video

- <track>
  - HTML element used as a child of the media elements - <audio> and <video>. It lets you specify timed text tracks (or time-based data), for example to automatically handle subtitles.
  - The tracks are formatted in WebVTT format (.vtt files) - Web Video Text Tracks.
  - Link: <https://developer.mozilla.org/en-US/docs/Web/HTML/Element/track>

# Web Video

- <source>
  - HTML element is used to specify multiple media resources for either the <picture>, the <audio> or the <video> element.
  - It is an empty element.
  - It is commonly used to serve the same media content in multiple formats supported by different browsers..
  - Link: <https://developer.mozilla.org/en-US/docs/Web/HTML/Element/source>

```
<video controls>
  <source src="foo.webm" type="video/webm">
  <source src="foo.ogg" type="video/ogg">
  I'm sorry; your browser doesn't support HTML5 video.
</video>
```

# Web Video

- recommended video formats:
  - webm – type = video/webm
  - mp4 – type = video/mp4
  - .ogg – type = video/ogg

```
<video controls>
  <source src="foo.webm" type="video/webm">
  <source src="foo.ogg" type="video/ogg">
I'm sorry; your browser doesn't support HTML5 video.
</video>
```

# Web Video

- Live Demo: <https://www.w3.org/2010/05/video/mediaevents.html>
- API: <https://www.w3.org/TR/html5/embedded-content-0.html#mediaevents>
- Media Player:
  - [https://developer.mozilla.org/en-US/Apps/Fundamentals/Audio and video delivery/Adding captions and subtitles to HTML5 video](https://developer.mozilla.org/en-US/Apps/Fundamentals/Audio_and_video_delivery/Adding_captions_and_subtitles_to_HTML5_video)

# Web Video

```
var W = canvas.width = video.clientWidth;
var H = canvas.height = video.clientHeight;

context.drawImage(video, 0, 0, W, H);
var imageData = context.getImageData(0, 0, W, H);

for (var y = 0; y < H; y++) {
    for (var x = 0; x < W; x++) {
        var i = (y * W * 4) + x * 4;
            // change values in imageData.data[i+...]
    }
}
context.putImageData(imageData, 0, 0);
// other canvas drawing operations
```

# Web Video

```
//create the video element using jQuery
var v = $("<video></video>")
    .attr({
        "controls": "",
        "autoplay": "",
        "src": "media/movie.mp4"
    })
    .load()
    .appendTo($("#body"));

// change the video file
v[0].src = "media/test.mp4";
v[0].load();
v[0].play();
```

# Web Video – Simple playlist

```
$ (function () {
    var lista = ["movie.mp4", "v2.mp4"];
    var index = 0;

    var video = $( "#myVideo" );

    video.on("ended", function () {
        index = index + 1;
        if (index >= lista.length) index = 0;

        video[0].src = lista[index];
        video[0].load();
        video[0].play();
    });
});
```

# Lab Examples

- <https://ase-multimedia.azurewebsites.net/video-player/>
- <https://ase-multimedia.azurewebsites.net/video-processing/>
- <https://ase-multimedia.azurewebsites.net/video-effects/>



# Audio

# Sound

- Sound is a form of mechanical energy transmitted as vibrations in a medium.
- The medium is usually air, though sound can also be transmitted through solids and liquids.
- Example: a clap of the hands produces sound by suddenly compressing and displacing air molecules. The disturbance is transmitted to adjacent molecules and propagated through space in the form of a wave. We hear the hand clap when these vibrations cause motions in the various parts of our ears.

# Sound

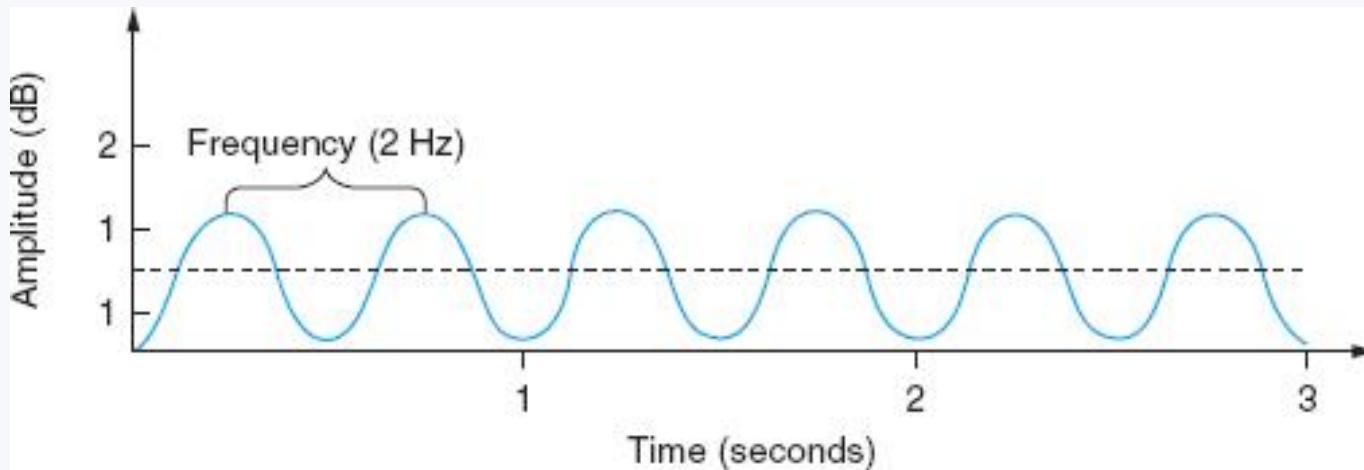
- Sound waves are often compared to the ripples produced when a stone is thrown into a pond. The stone displaces the water to produce high points, or wave peaks, and low points, or troughs. We see these alternating peaks and troughs as patterns of waves moving outward from the stone's point of impact. The clap of the hand also produces peaks and troughs as high air pressure is followed by a return to lower pressure. These changes in air pressure produce patterns of waves spreading in all directions from the sound's source.



# Sound

- Humans can hear sound waves with frequencies between about 20 Hz and 20 kHz
- **Noise** is unwanted sound judged to be unpleasant, loud or disruptive to hearing. From a physics standpoint, noise is indistinguishable from sound, as both are vibrations through a medium, such as air or water. The difference arises when the brain receives and perceives a sound.

# Representation

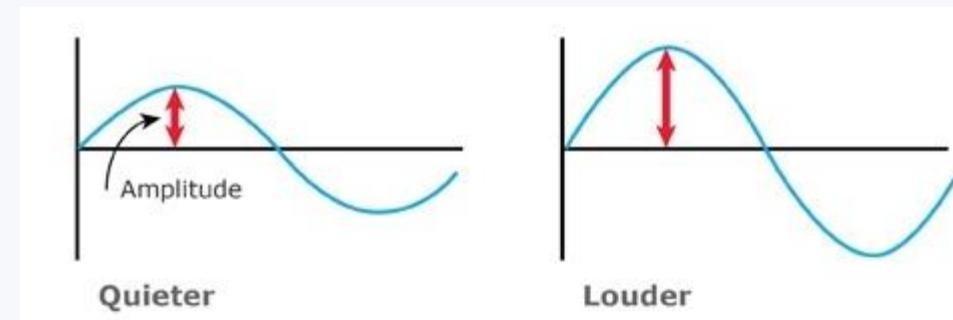


# Characteristics

- amplitude
- frequency
- duration

# Characteristics

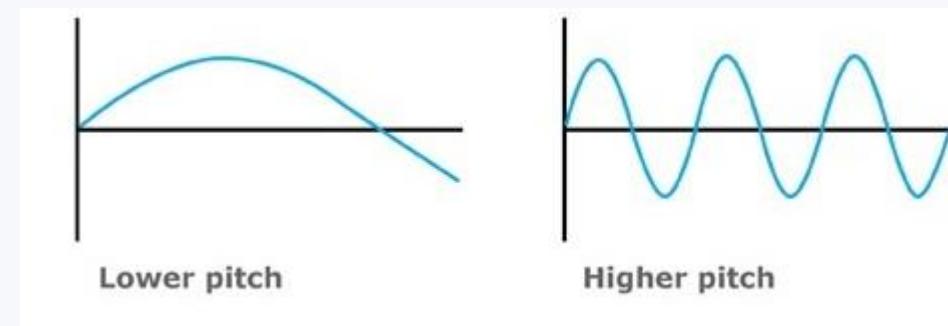
- **Amplitude** is a measure of sound pressure or the amount of energy associated with the sound. This is represented by the vertical, or *y*-axis, of the sine wave. Amplitude is perceived as the sound's **volume**, which is usually measured in **decibels (dB)**.



- In general, sounds with higher amplitudes are experienced as *louder*. The range of human hearing is approximately 3 to 140 dB. Each 10 dB increase roughly doubles the perceived volume of a sound.

# Characteristics

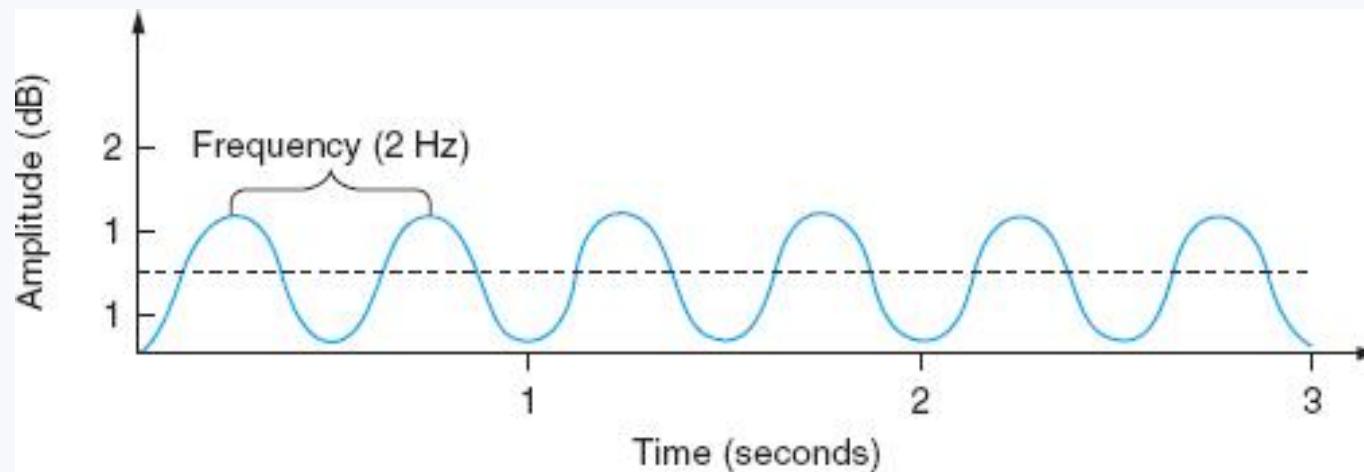
- **Frequency** is the number of times a waveform repeats in a given interval of time. It is represented on the horizontal axis as the distance between two wave peaks or troughs. Frequency is measured in *hertz* (Hz).
- One **hertz** is one repetition of a waveform in one second of time.



- Frequency is perceived as **pitch**. High frequencies produce sounds of higher pitch, and low frequencies produce low pitch. Pitch is the psychological perception of sound frequency. Humans can perceive a frequency range of 20 Hz to 20,000 Hz (or 20 *kilohertz* (kHz), *thousands* of hertz), though most adults cannot hear frequencies above 16 kHz.

# Characteristics

- The **duration** of the sound is the length of time it lasts. The total length of the horizontal axis represents duration

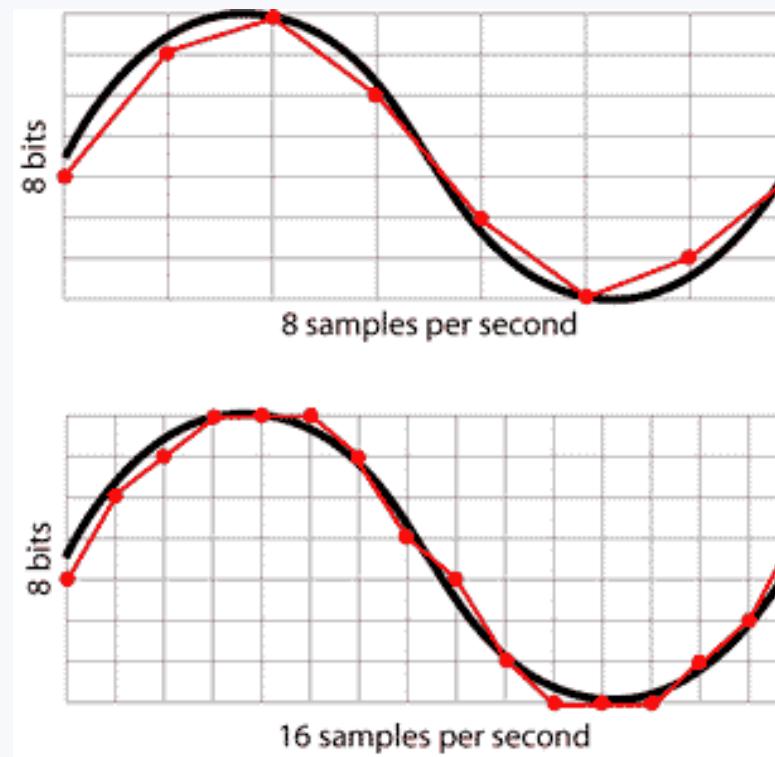


# Digital Audio

- Digital techniques represent sound as discrete (or discontinuous) elements of information.
- There are two major types of digital sound:
  - *sampled*
  - *synthesized*.

# Digital Audio

- Sampled sound is a digital recording of previously existing analog sound waves. A file for sampled sound contains many thousands of numerical values, each of which is a record of the amplitude of the sound wave at a particular instant, a *sampling* of the sound.



# Digital Audio

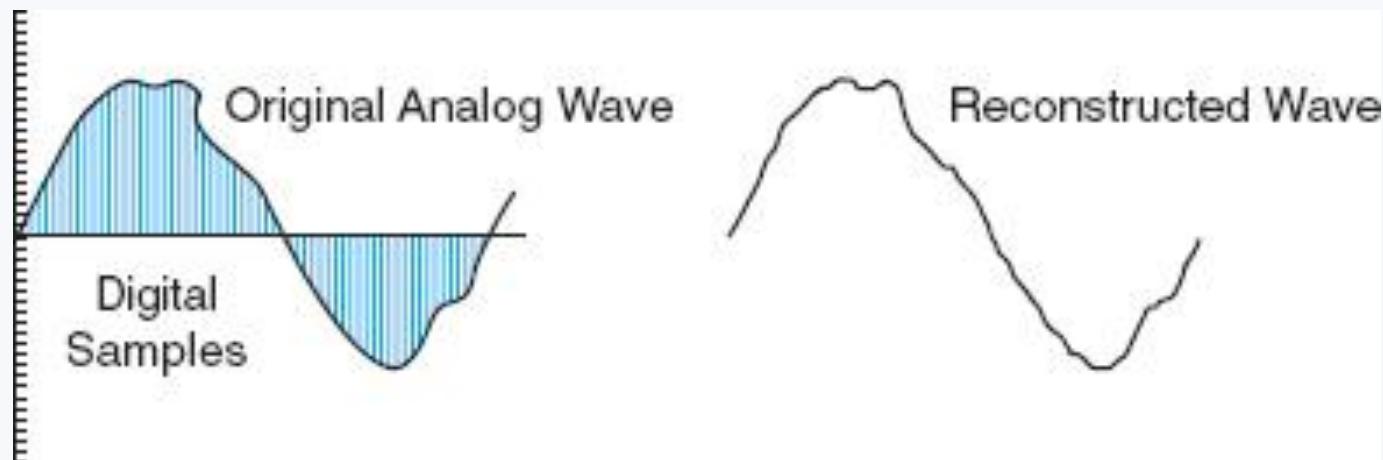
- **Synthesized** sound is new sound generated (or synthesized, “put together”) by the computer. A file for synthesized sound contains instructions that the computer uses to produce its own sound.
- **Sampling** is usually used to capture and edit naturally occurring sounds such as human speech, musical and dramatic performances, bird calls, rocket launches, and so on. **Synthesized** sound is generally used to create original musical compositions or to produce novel sound effects.

# Digital sampling

- capturing and storing sound in a digital format.
- the sound is captured by recording many separate measurements of the amplitude of a wave using an **ADC**, or *analog-to-digital converter*. An analog device, such as a microphone or the amplifier in a speaker system, generates a continuously varying voltage pattern to match the original sound wave. The ADC samples these voltages thousands of times each second. The samples are recorded as digital numbers. These digital values are then used to re-create the original sound by converting the digital information back to an analog form using a **DAC**, or *digital-to-analog converter*. The DAC uses the amplitude values to generate matching voltages that power speakers to reproduce the sound.

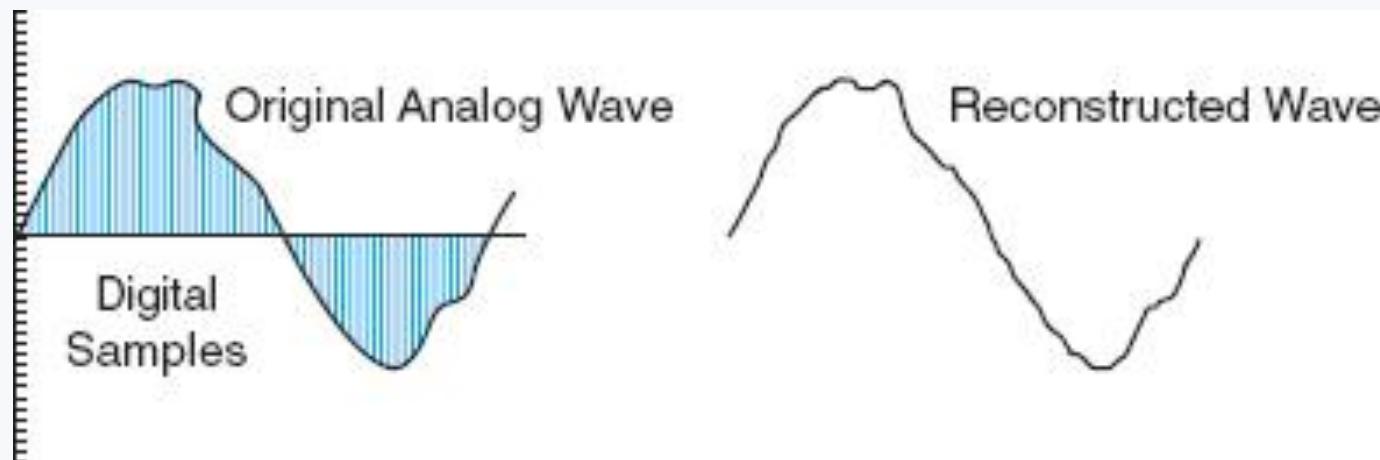
# Digital sampling

- Digital sampling replaces the continuous waveform of the original sound with a new wave created from a fixed number of discrete samples.
- Some information is always lost in sampling, because a continuous wave is infinitely divisible and sampling always yields a finite number of values.
- The quality of sampled sound is dependent on two factors directly connected to this sampling process: **sample resolution** and **sample rate**.



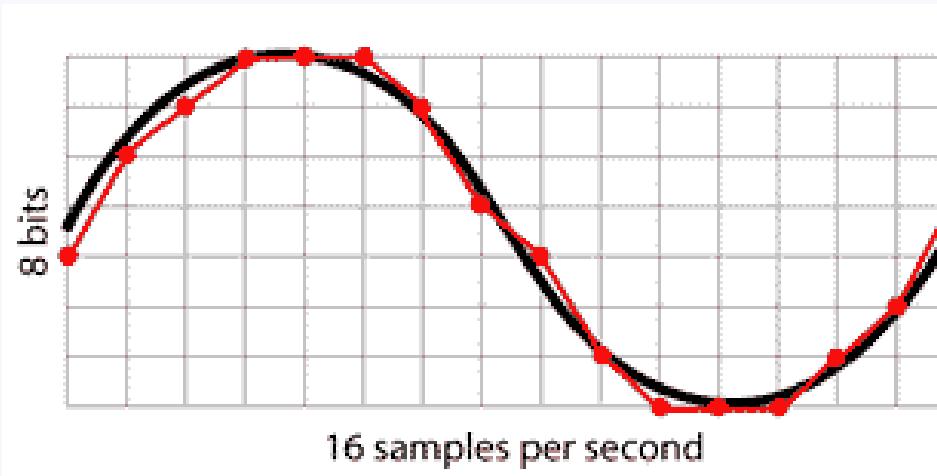
# Digital sampling

- Digital sampling replaces the continuous waveform of the original sound with a new wave created from a fixed number of discrete samples.
- Some information is always lost in sampling, because a continuous wave is infinitely divisible and sampling always yields a finite number of values.
- The quality of sampled sound is dependent on two factors directly connected to this sampling process: **sample resolution** and **sample rate**.



# Digital sampling

- **Sample Resolution** Each measurement of amplitude made by an ADC is recorded using a fixed number of bits. The number of bits used to encode amplitude is known as **sample resolution**.
- Sample resolutions for digital audio range from 8 to 32 bits, with the most common being the 16-bit CD-Audio standards and 24-bit DVD-Audio standards.



## Digital sampling

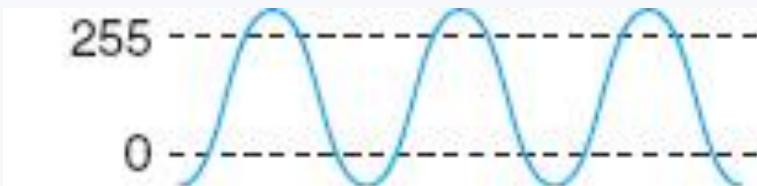
- Eight bits can record 256 different amplitude levels. This is adequate to capture the variations in limited decibel ranges, such as those between a human whisper and a shout, but higher sample resolutions are needed to accurately reproduce sounds with a wider range of amplitudes, such as musical performances. Thus, 8-bit audio is generally used only for simple sounds or in multimedia applications requiring very small file sizes.
- The CD-Audio standard, with its 16 bits per sample, supports over 65 thousand different amplitude levels, whereas the 24-bit DVD-Audio standard can represent over 16 million levels.

# Digital sampling

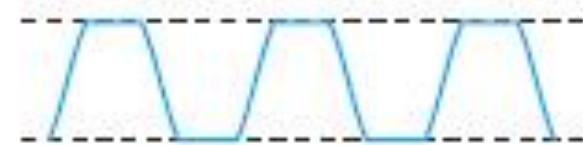
- Inadequate sample resolution can distort sound in two different ways: **quantization** and **clipping**.
- **Quantization** - each amplitude sample must be assigned one of the numbers available in the code being used. If the number of distinct values is too small, perceptually different amplitudes will be assigned the same number. Rounding a sample to the closest available value is known as **quantization**. In the case of sound, excessive quantization may produce a background hissing or a grainy sound. The solution is to record with a higher sample resolution (for instance, by using 16 rather than 8 bits).

# Digital sampling

- **Clipping** - sound-sampling equipment is designed for a selected decibel range. If the source sound exceeds this range (as, for instance, when someone yells into a microphone held close to their lips), higher amplitudes cannot be encoded, because no values are available to represent them. The waveform of a clipped sound shows square tops and bottoms marking the point at which the highest amplitudes could not be captured. Clipping can produce a harsh, distorted sound.



Clipping occurs when wave amplitude exceeds available sample values.



Clipped waves have square tops and bottoms and a harsh sound.

# Digital sampling

- The solution to **clipping** is to lower the amplitude of the source sound to record within the limits of the ADC circuitry.
- Recording equipment usually includes some form of meter such as a swinging needle, colored bars, or lights to show input levels and alert users when the amplitude range has been exceeded.
- The familiar, “Testing—one, two, three,” is often used to establish the proper distance and speech level when recording with a microphone.

# Digital sampling

- Clipping can also occur during the mixing of different audio tracks.
- **Mixing** is the process of combining two or more sound selections, or *tracks*, into a single track. For example, a background music track might be mixed with a voice track of a poetry reading. This combination of two or more tracks may produce an amplitude that exceeds the available range. Adjustments to the volume of each track can eliminate the problem.
- Another solution is to use higher sample resolutions (for instance, 24-bit) to provide a wider range of amplitude values.

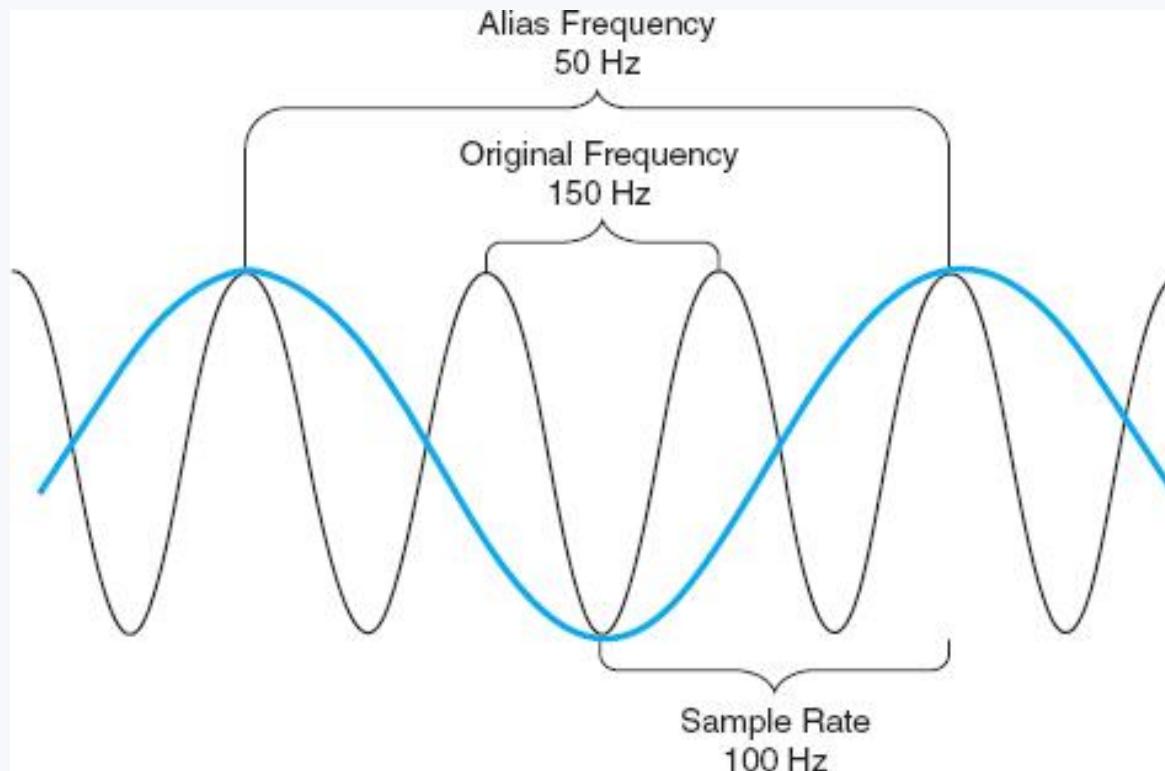
# Digital sampling

- **Sample rate** is the number of samples taken in a fixed interval of time. A rate of one sample per second is designated as a Hertz.
- Because sound samples are always taken thousands of times each second, sample rates are usually stated in kilohertz.
- Sample rate affects sound quality by determining the range of frequencies that can be represented in a digital recording. At least two measurements are required to capture each cycle of a sound wave—one for each high value, or peak, and one for each low value, or trough. The highest frequency that can be captured is thus one-half of the sample rate.
- CD-quality sound captures 44,100 samples per second (44.1 kHz sample rate) and can represent frequencies as high as 22,050 Hz or 22.05 kHz. DVD-Audio uses a sampling frequency of 96 kHz to capture frequencies as high as 48 kHz.

## Digital sampling

- Sounds that do not contain high frequencies can be more efficiently represented using lower sample rates because this will produce a smaller overall file size. One potential problem with lower sample rates, however, is **aliasing**.

# Digital sampling



# Balancing File Size and Sound Quality

- It is not always necessary to use DVD-quality sound. For example normal speech, contain relatively low frequencies and a limited range of amplitudes. In this case, higher sample rates and sample resolutions do not improve sound quality, but they create needlessly large files. Using a rate of 11.025 kHz for voice recording results in a file only one-quarter the size of a CD-quality sound file. In addition, 8-bit sample resolution and monaural sound are usually adequate for speech. This further reduces file size by a factor of four, one-sixteenth the size of a stereo CD recording.

Resolution	Rate	Stereo/Mono	Size (1 Minute)	Quality
16	44.1 kHz	Stereo	10 MB	CD
16	22.05 kHz	Stereo	5 MB	FM radio
8	11.025 kHz	Mono	650 KB	AM radio
8	5.5 kHz	Mono	325 KB	Bad telephone

# Sound Compression

- Lowering the sample rate and reducing sample resolution are two ways to reduce the size of a sampled sound file. These methods work well for sounds at relatively low frequencies and narrow amplitude ranges. They are not effective for sounds that contain wider ranges of both frequency and amplitude, such as musical performances. In these cases another strategy can be used: compression.
- Compression can be either **lossless** or **lossy**. Lossless compression uses more efficient coding to reduce the size of a file while preserving all the information of the original. Lossy compression discards some of the original information.

# Sound Compression

- Because **lossy strategies** produce much smaller files, they are the preferred technique for sound compression.
- Lossy sound compression **codecs** (coder/decoders) use various techniques to reduce file sizes. Some of these take advantage of **psychoacoustics**, the interplay between the psychological conditions of human perception and the properties of sound. For instance, while humans with optimal hearing can perceive frequencies as high as about 20 kHz, most people cannot distinguish frequencies above approximately 16 kHz. This means that higher-frequency information can be eliminated and most listeners will not miss it. Higher amplitude sounds in one stereo channel will also typically “drown out” softer sounds in the other channel. Again, this is information that usually will not be missed.

# Sound Compression

- Lossy compression also uses other techniques such as **variable bit rate encoding (VBR)**. In VBR, sounds are encoded using a different number of bits per second depending on the complexity of the sound. For simple passages of sound with limited frequencies, a smaller number of bits per second is used than for more complex passages, such as those with many different instruments and higher frequencies.
- Lossy codecs such as the widely supported **MP3** can reduce file sizes by as much as 80% while remaining virtually indistinguishable from the original CD-quality sound.

# Sampled Sound File Formats

- MP3 (MPEG1, audio layer 3)
  - extensions: .mp3 or .mpga
  - popular audio format that supports significant compression while preserving excellent quality.
  - widely supported across different computer platforms and is frequently used on the Internet.

# Sampled Sound File Formats

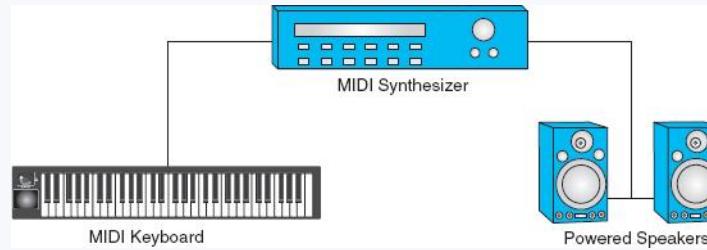
- AAC (advanced audio coding)
  - extension: .mp4 and .aac
  - the successor to MP3 specified in the MPEG4 standard, that produces better sound quality than MP3 at comparable bit rates.
  - it can also significantly reduce file sizes for comparable-quality audio.
  - many commercial users (including Apple iPod, Apple iPad, Apple iPhone, Blackberry, YouTube, and Sony PlayStation) have adopted the AAC standard for their digital audio.

# Sampled Sound File Formats

- Other sound file formats:
  - WMA (Windows Media Audio);
  - WAV;
  - ReadAudio;
  - AIFF;
  - AU.

## Audio

# Synthesized Sound



Further reading: <https://en.wikipedia.org/wiki/MIDI>

# Web Audio

- <audio> - HTML element is used to embed sound content in documents. It may contain one or more audio sources, represented using the src attribute or the <source> element; the browser will choose the most suitable one.

```
<audio controls>
  <source src="foo.wav" type="audio/wav">
  Your browser does not support the <code>audio</code>
  element.
</audio>
```

# Web Audio

- <audio>
  - attributes:
    - *autoplay*
    - *controls*
    - *loop*
    - *src*
    - *volume* (numeric) – value between 0 and 1
  - API: <https://developer.mozilla.org/en-US/docs/Web/HTML/Element/audio>

# Web Audio

- `HTMLAudioElement` (JavaScript)
  - provides access to the properties of `<audio>` elements, as well as methods to manipulate them. It derives from the `HTMLMediaElement` interface.
  - properties:
    - `currentSrc`, `currentTime`, `duration`, `ended`, `error`, `paused`, `readyState`, `volume`
  - methods:
    - `canPlayType`, `load`, `pause`, `play`
  - events:
    - `canplay`, `ended`, `pause`, `play`, `volumechange`, `waiting`
  - API: <https://developer.mozilla.org/en-US/docs/Web/API/HTMLAudioElement>

## Lab Example:

- <https://ase-multimedia.azurewebsites.net/audio-playlist/>



# Web Audio API

- The Web Audio API provides a powerful and versatile system for controlling audio on the Web, allowing developers to choose audio sources, add effects to audio, create audio visualizations, apply spatial effects (such as panning) and much more.
- API: [https://developer.mozilla.org/en-US/docs/Web/API/Web\\_Audio\\_API](https://developer.mozilla.org/en-US/docs/Web/API/Web_Audio_API)
- Web RTC API: <https://developer.mozilla.org/en-US/docs/Web/API/MediaDevices/getUserMedia>

# Web Audio

- Sound visualization using HTML5 Canvas: <http://nipe-systems.de/webapps/html5-web-audio/>
- Creating visualizations: [https://developer.mozilla.org/en-US/docs/Web/API/Web\\_Audio\\_API/Visualizations\\_with\\_Web\\_Audio\\_API](https://developer.mozilla.org/en-US/docs/Web/API/Web_Audio_API/Visualizations_with_Web_Audio_API)
- Record and save as MP3: <http://audior.ec/blog/recording-mp3-using-only-html5-and-javascript-recordmp3-js/> ,  
<https://www.html5rocks.com/en/tutorials/getusermedia/intro/>

# WebRTC API

- WebRTC (Web Real-Time Communications) is a technology which enables Web applications and sites to capture and optionally stream audio and/or video media, as well as to exchange arbitrary data between browsers without requiring an intermediary.
- The set of standards that comprises WebRTC makes it possible to share data and perform teleconferencing peer-to-peer, without requiring that the user install plug-ins or any other third-party software.
- Web RTC API: [https://developer.mozilla.org/en-US/docs/Web/API/WebRTC\\_API](https://developer.mozilla.org/en-US/docs/Web/API/WebRTC_API),  
<https://developer.mozilla.org/en-US/docs/Web/API/MediaDevices/getUserMedia>

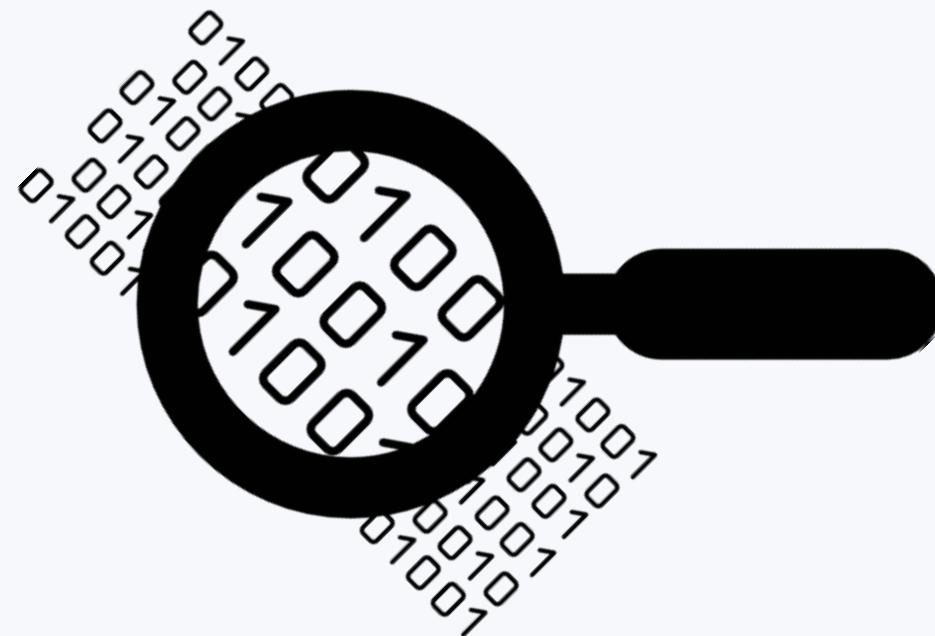
# Demo

- [https://developer.mozilla.org/en-US/docs/Web/API/WebRTC\\_API/Taking\\_still\\_photos](https://developer.mozilla.org/en-US/docs/Web/API/WebRTC_API/Taking_still_photos)



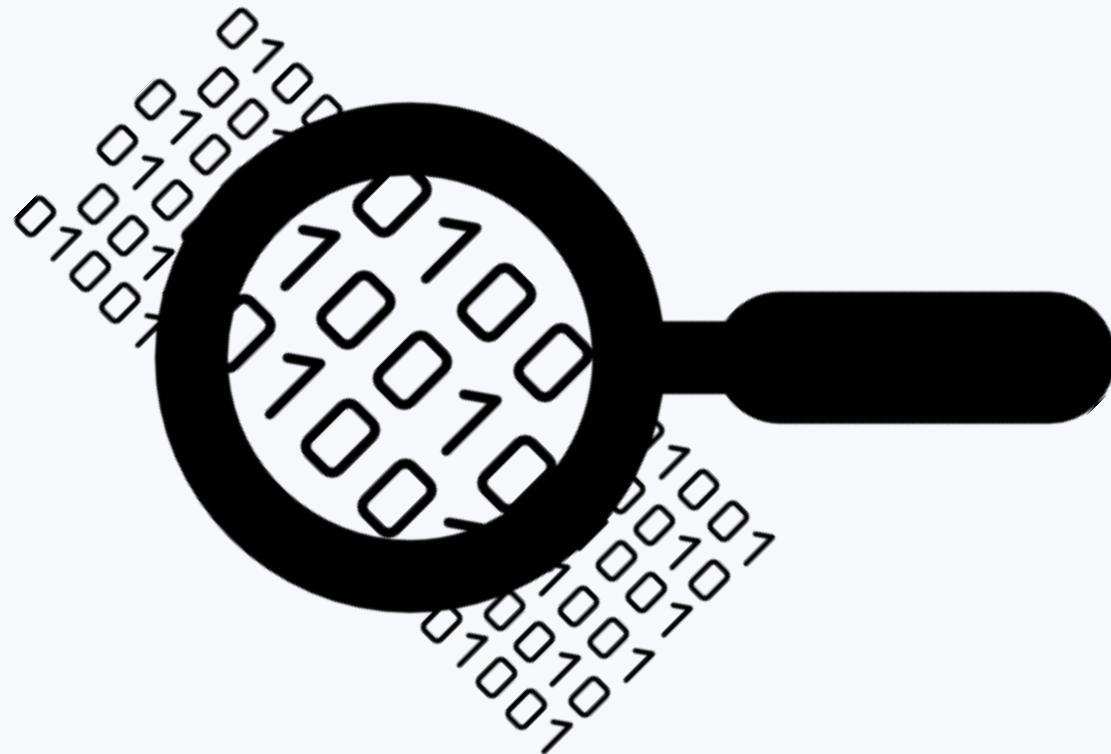
# Lab Example: Web Audio API

- <https://ase-multimedia.azurewebsites.net/audio-web-audio-api/>



# Lab Example: Speech API

- <https://ase-multimedia.azurewebsites.net/speech-api/>



# Recommendations

- Consider the playback environment:
  - Public vs. private use (ex: autoplay)
  - Give users control:
    - Over volume.
    - To stop or start play.
- Avoid excessive use of sound. Sound can be more tiring for users than images or text

# Animation

# Animation

- Animation is the process of creating the illusion of motion and the illusion of change
- Main techniques:
  - movie technique
  - key frames
  - color changing

# Digital Animation

- Digital animation takes two different forms:
- two-dimensional - has evolved from traditional techniques, particularly cel animation;
- three-dimensional - exploits the capabilities unique to the computer to produce an entirely new form of animation.

# Digital Animation

- 1. frame-by-frame animation
  - animators produce each successive frame manually
  - technique that provides complete control over frame content, but is also very time consuming

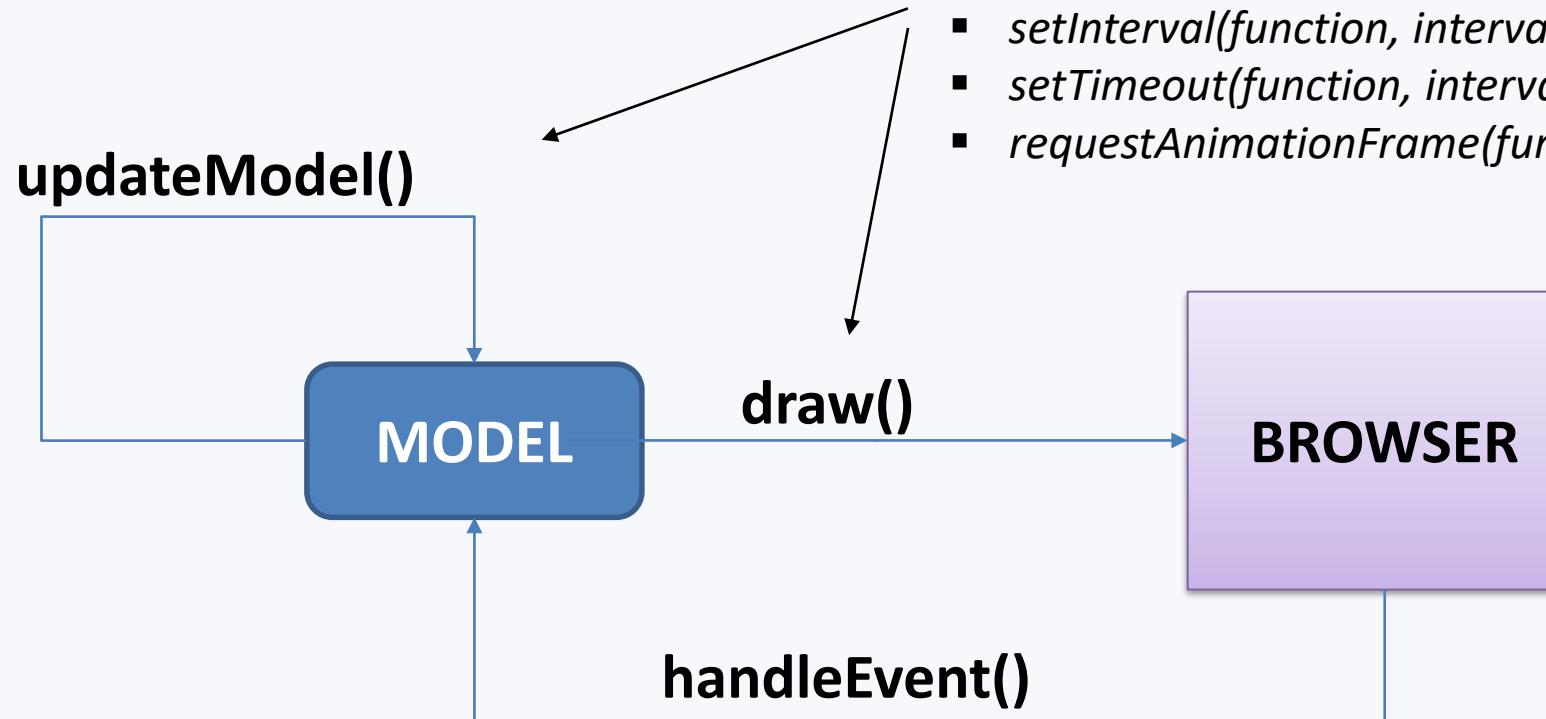
# Digital Animation

- 2. tween animation
  - the animator creates the key frames and the computer automatically produces the tweens.
  - There are several different types of tweens:
    - motion tween - can be used to move an object from one position to another. The first key frame places the object in one position and the second places it in another. The program then fills in the intervening frames.
    - path-based animation: the animator draws a path from one key frame to another and the computer fills in the intervening images, spaced out along the path, to produce the desired motion
    - morphing - the shape of one image is gradually modified until it changes into another shape

# Digital Animation

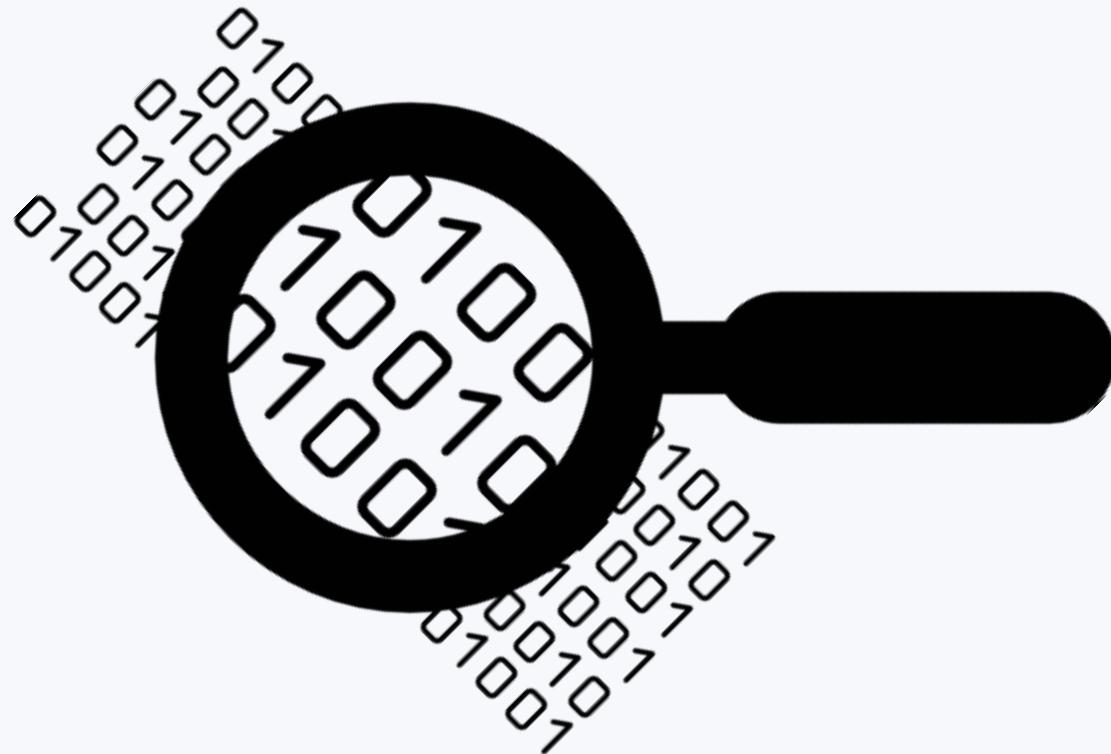
- size tweening - the first key frame is the object at its initial size and the second is the final size.
- alpha tweening- color and transparency can be animated using key frames representing initial and final image properties.
- 3. programmed animation

# Web Animation



# Demo

- lab animation examples



# CSS Transitions

- Guide: [https://developer.mozilla.org/en-US/docs/Web/CSS/CSS\\_Transitions/Using\\_CSS\\_transitions](https://developer.mozilla.org/en-US/docs/Web/CSS/CSS_Transitions/Using_CSS_transitions)

# CSS Animations

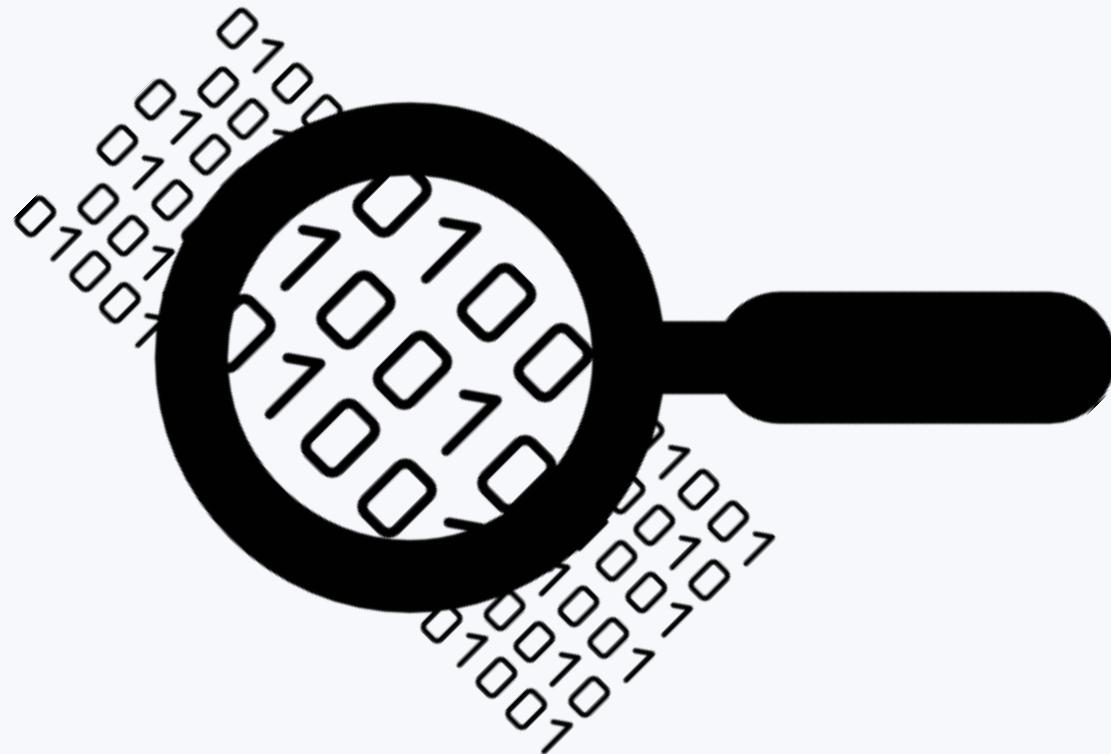
- API: [https://developer.mozilla.org/en-US/docs/Web/CSS/CSS\\_Animations](https://developer.mozilla.org/en-US/docs/Web/CSS/CSS_Animations)
- Guide: [https://developer.mozilla.org/en-US/docs/Web/CSS/CSS\\_Animations/Using\\_CSS\\_animations](https://developer.mozilla.org/en-US/docs/Web/CSS/CSS_Animations/Using_CSS_animations)

# Web Animations API

- The **Web Animations API** allows for synchronizing and timing changes to the presentation of a Web page, i.e. animation of DOM elements.
- Limited browser support: <https://caniuse.com/#feat=web-animation> (compared to **CSS Animations** with <https://caniuse.com/#feat=css-animation> )
- API: [https://developer.mozilla.org/en-US/docs/Web/API/Web\\_Animations\\_API](https://developer.mozilla.org/en-US/docs/Web/API/Web_Animations_API)

## Demo

<https://css-tricks.com/css-animations-vs-web-animations-api/>



# Progressive Web Apps

# Progressive Web Apps

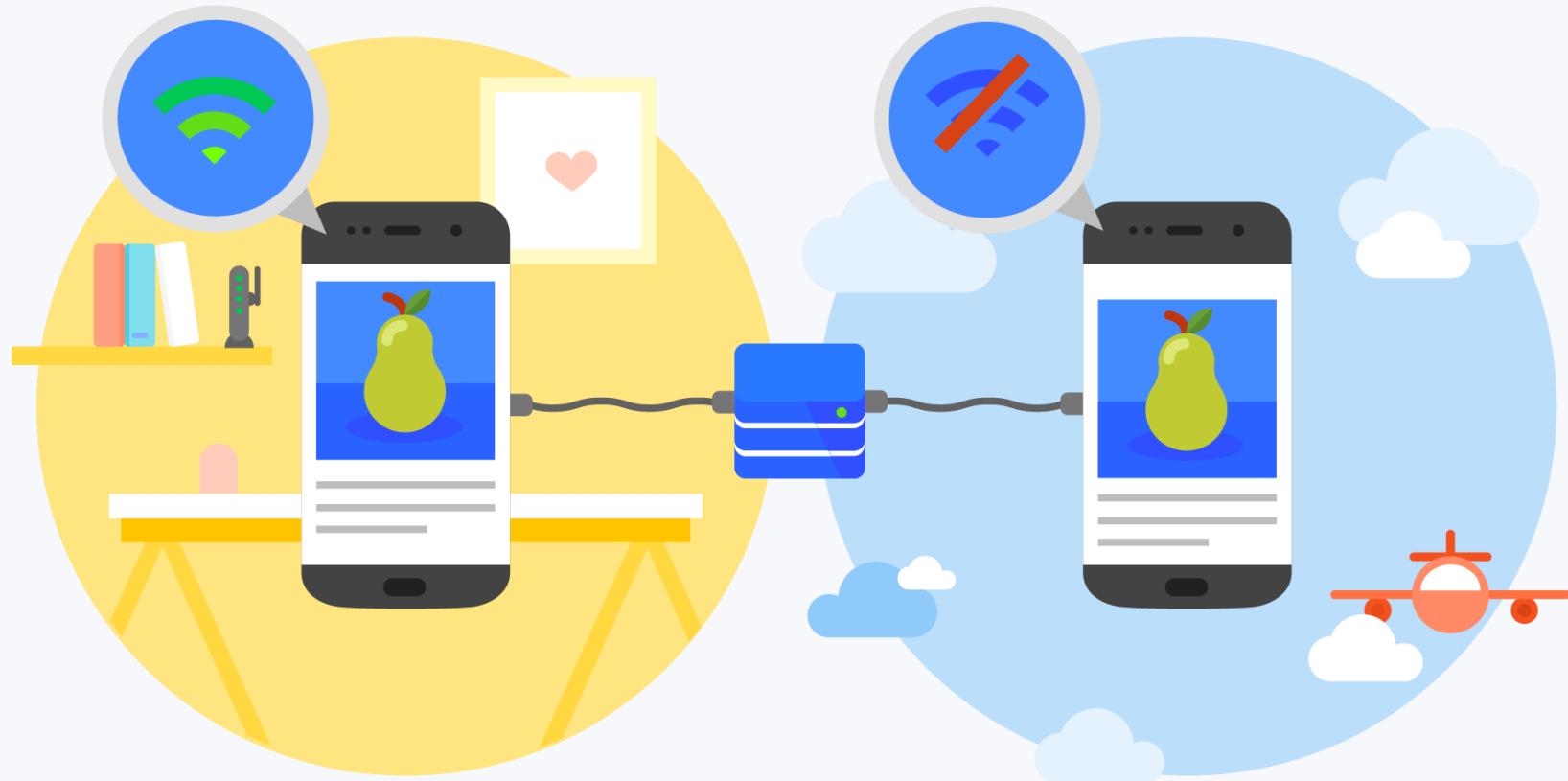
- An application developed using standard web technologies (HTML5, JavaScript, CSS3) which is:
- **Reliable** - load instantly even in uncertain network conditions.
- **Fast** - respond quickly to user interactions with silky smooth animations and no janky scrolling.
- **Engaging** - feel like a natural app on the device, with an immersive user experience.

# Reliable

- Progressive web apps are not dependent on the user's connection like traditional websites are.
- When a user visits a progressive web app, it will register a **service worker** that can detect and react to changes in the user's connection. It can provide a fully featured user experience for users who are offline, online, or suffering from an unreliable connection.
- A **service worker**, written in JavaScript, is like a **client-side proxy** and puts you in control of the cache and how to respond to resource requests. By pre-caching key resources you can eliminate the dependence on the network, ensuring an instant and reliable experience for your users.

## Reliable

- When launched from the user's home screen, **service workers** enable a Progressive Web App to load instantly, regardless of the network state.



# Reliable

- A service worker, written in JavaScript, is like a client-side proxy and puts you in control of the cache and how to respond to resource requests. By pre-caching key resources you can eliminate the dependence on the network, ensuring an instant and reliable experience for your users.

### Fast

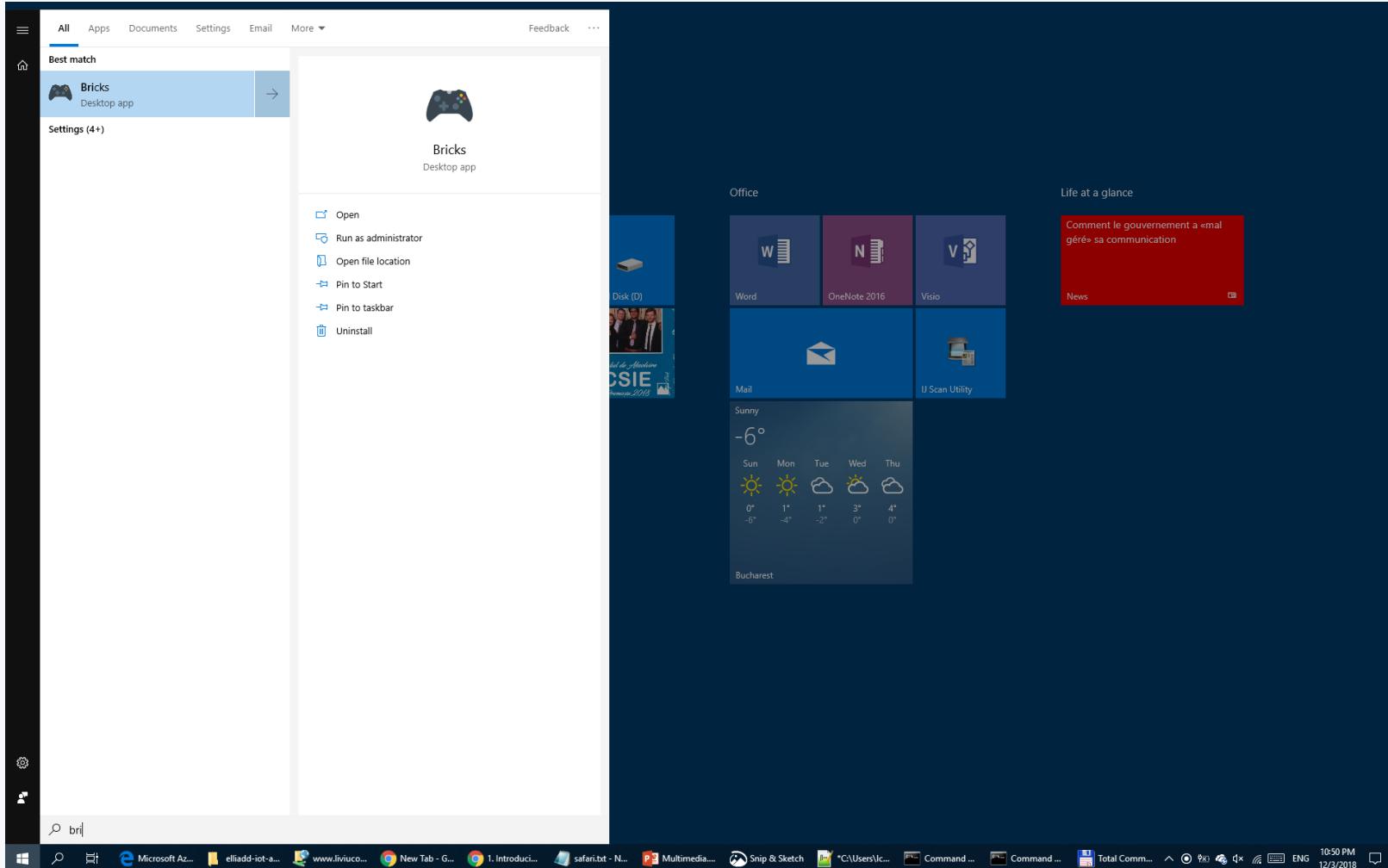
- Using service workers, progressive web apps can launch in an instant, whether the user has a blazing fast connection, an unreliable 2G connection, or even no connection at all. Sites can load in milliseconds, much faster than anything we have experienced on the web before, and sometimes faster than native apps.

# Engaging

- Push notifications
  - Progressive web apps can send notifications to their users. The notifications have a completely native feel and are indistinguishable from native app notifications.
- Homescreen shortcut
  - Once a user has shown interest in a progressive web app, the browser will automatically suggest that he add a shortcut to his homescreen—completely indistinguishable from any native app.

# Progressive Web Apps

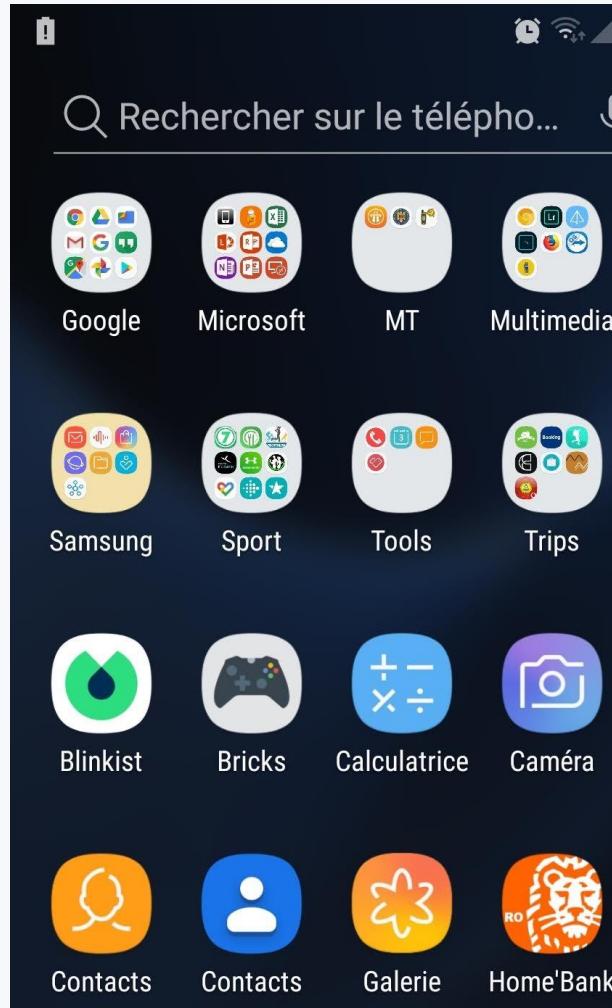
## Engaging



Desktop - Homescreen shortcut

# Progressive Web Apps

## Engaging



Mobile - Homescreen shortcut

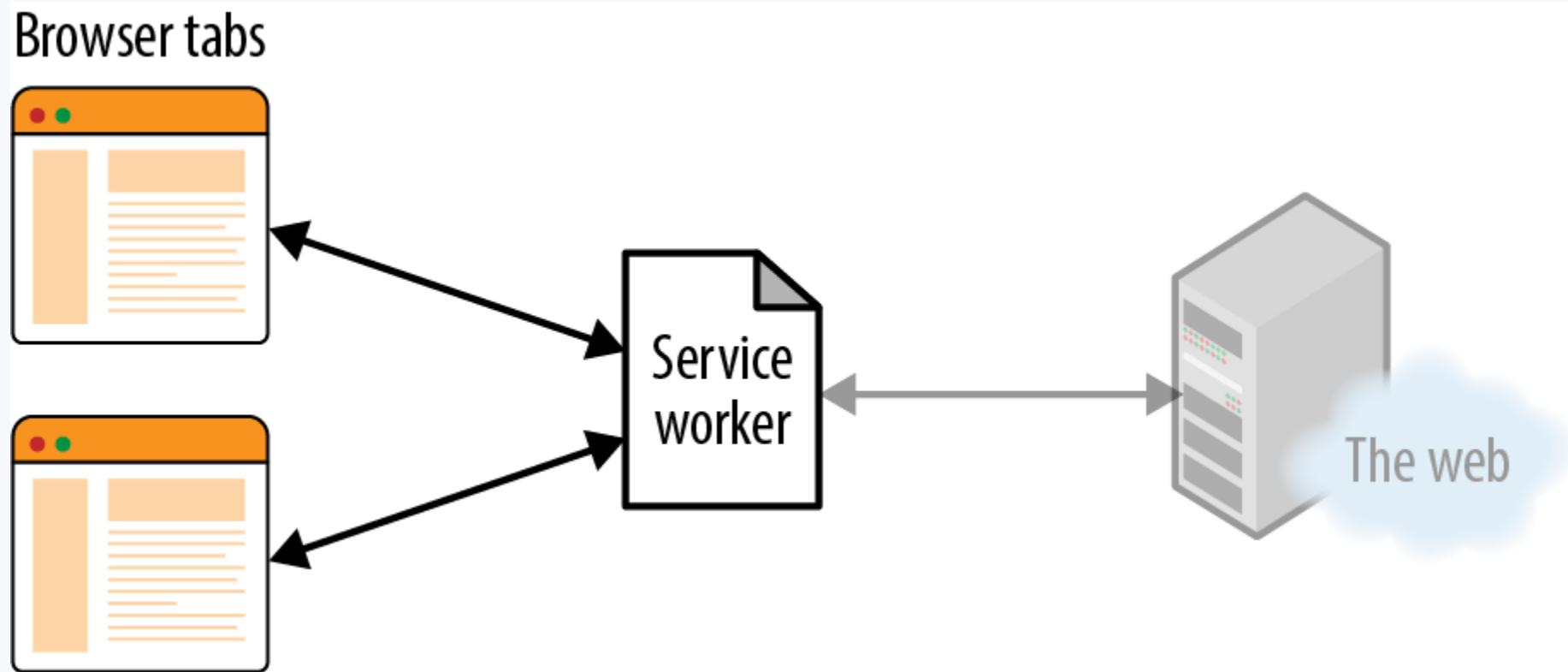
# Engaging

- Native look
  - Progressive web apps launched from the homescreen can have an entirely native, app-like look. They can have a splash screen as they are loading. They can launch in full-screen mode, without the browser and phone UI around them. They can even lock themselves to a specific screen orientation (a vital requirement for games).

# Service Worker

- At the heart of every progressive web app is the service worker.
- Before service workers, we had code running either on the server or in the browser window. Service workers introduce another layer.
- A service worker is a script that can be registered to control one or more pages of your site. Once installed, a service worker sits outside of any single browser window or tab.
- From this place, a service worker can listen and act on events from all pages under its control. Events such as requests for files from the web can be intercepted, modified, passed on, and returned to the page

# Service Worker



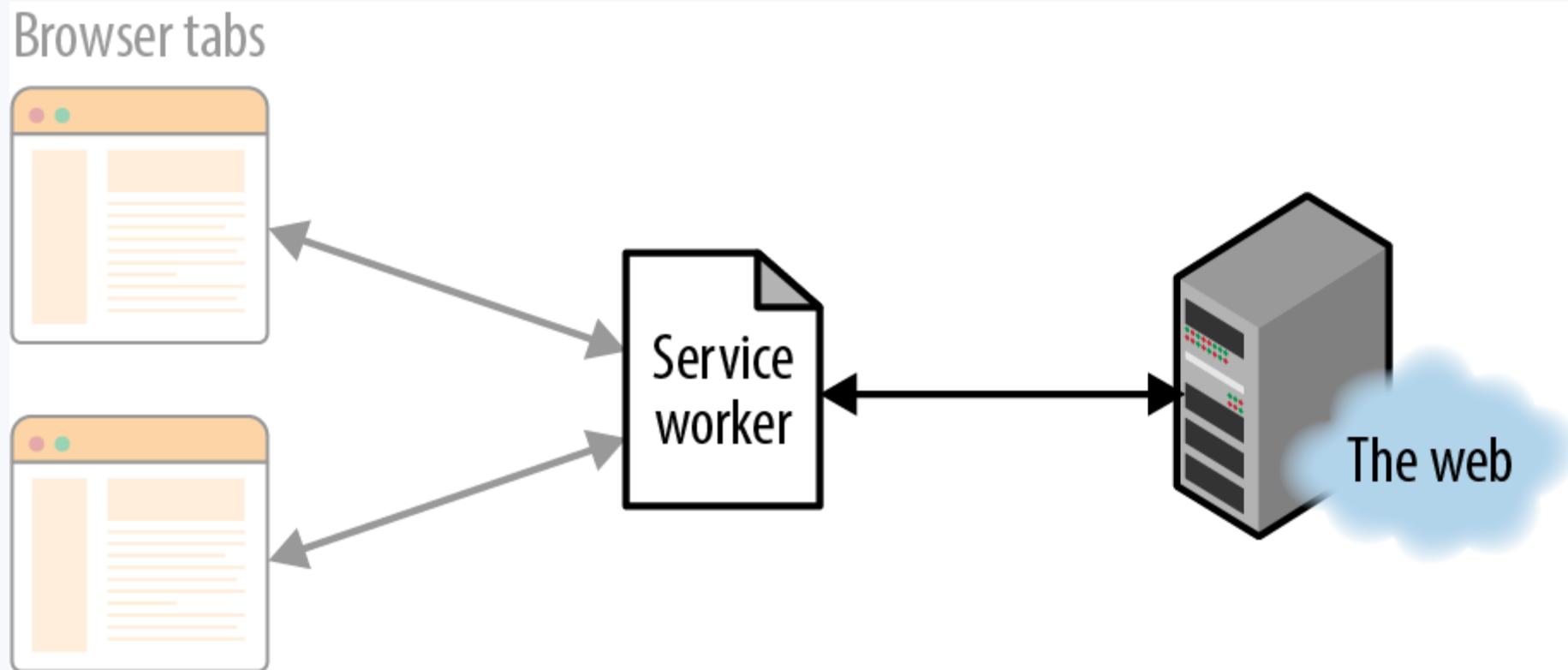
Pages communicating with a service worker while the user is **offline**

# Checking the tabs associated with a service worker

- Using the Developer Tools in Google Chrome we can easily check the browser tabs corresponding to a service worker

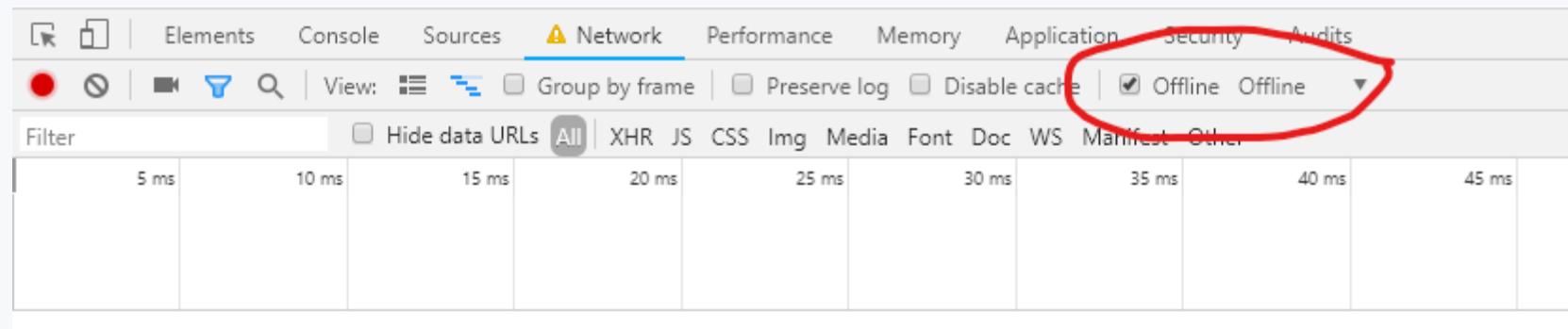
The screenshot shows the 'Service Workers' panel in the Google Chrome DevTools. At the top, there are three checkboxes: 'Offline', 'Update on reload', and 'Bypass for network'. Below this, the URL '127.0.0.1' is displayed. Under 'Source', a link to 'serviceworker.js' is shown with a red '8' indicating multiple errors. The 'Received' timestamp is '12/11/2018, 11:40:44 AM'. The 'Status' section shows two entries: '#5335 activated and is running' (green dot) and '#5501 waiting to activate' (orange dot). The 'Received' timestamp for the second entry is '12/17/2018, 10:49:51 PM'. The 'Clients' section lists three clients, each with a yellow 'focus' status indicator: 'http://127.0.0.1:5500/index.html' (repeated twice).

# Service Worker



The service worker communicating with a server after a user has left the page

# Simulating an offline state in Google Chrome



# Service Worker

- Documentation:
  - [https://developer.mozilla.org/en-US/docs/Web/API/Service Worker API](https://developer.mozilla.org/en-US/docs/Web/API/Service_Worker_API)
- A service worker is run in a worker context: it therefore has no DOM access, and runs on a different thread to the main JavaScript that powers your app, so it is not blocking.
- It is designed to be fully async; as a consequence, APIs such as synchronous [XMLHttpRequest](#) and [localStorage](#) can't be used inside a service worker.
- Service workers only run over HTTPS, for security reasons.

# Service Worker

- Registration:
  - A service worker is first registered using the `ServiceWorkerContainer.register()` method. If successful, the service worker will be downloaded to the client and attempt installation/activation for URLs accessed by the user inside the whole origin, or inside a subset specified by you.
- Documentation:
  - <https://developer.mozilla.org/en-US/docs/Web/API/ServiceWorkerContainer/register>

# Service Worker

- Note: check the [documentation](#) for an example also declaring a scope

```
if ("serviceWorker" in navigator) {
  navigator.serviceWorker.register("/serviceworker.js")
    .then(function(registration) {
      console.log("Service Worker registered with scope:", registration.scope);
    }).catch(function(err) {
      console.log("Service worker registration failed:", err);
    });
}
```

# Demo

- Create an empty html page and try to register a service worker
  - Note: the registration will fail because we have not yet created the "serviceworker.js" file

```
Service worker registration failed: TypeError: Failed to register a index.html:20
ServiceWorker: A bad HTTP response code (404) was received when fetching the script.
```



# Service Worker

- Verify that the current browser supports service workers

```
if ("serviceWorker" in navigator) {
```

- The register call returns a *promise*. If the promise is fulfilled, meaning the service worker was registered successfully, the function defined in the then statement is called. If there was any problem, the function defined inside the catch block would be executed.

```
navigator.serviceWorker.register("/serviceworker.js")
```

- If there was any problem, the function defined inside the catch block would be executed.

# Service Worker

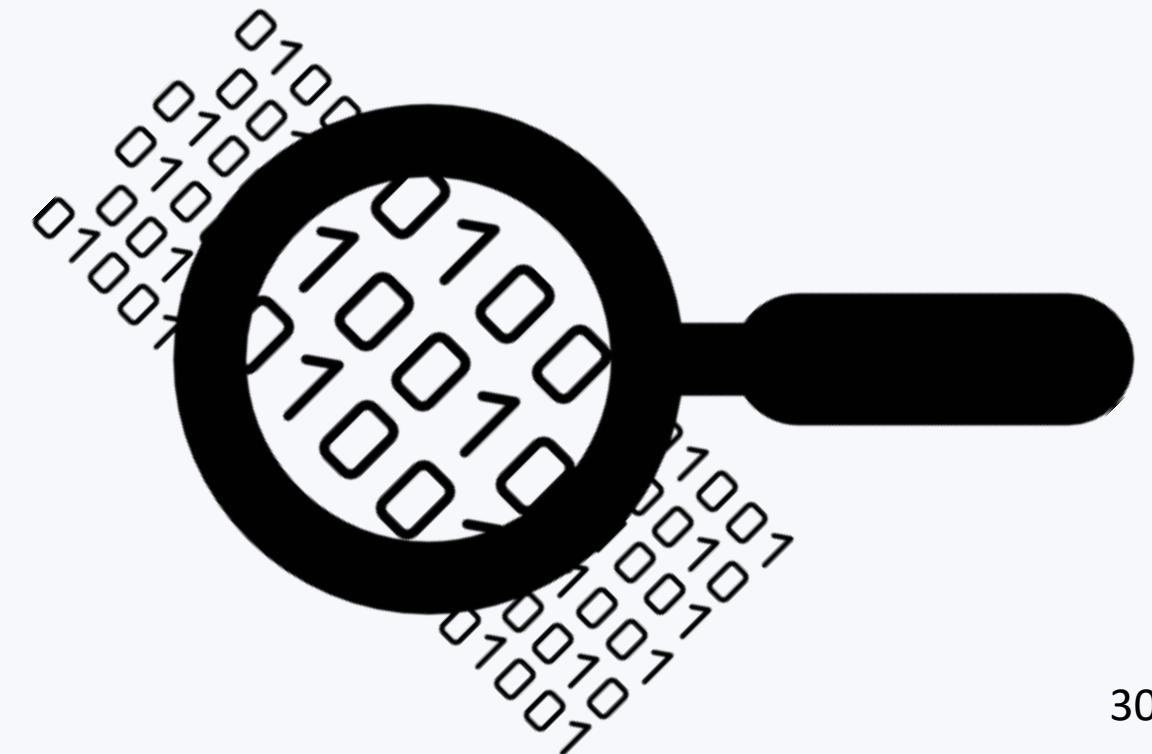
- Intercept requests from the Service Worker

```
self.addEventListener("fetch", function(event) {  
    console.log("Fetch request for:",  
    event.request.url);  
});
```

- The code adds an event listener to our service worker by calling addEventListener on self (self within a service worker refers to the service worker itself).
- The function we define to handle these events accesses the request object (available as a property of the fetch event) and logs the URL of that request.

# Demo

- Create a service worker and subscribe to the “fetch” event



# Service Worker

- every request made by our page (including to third-party servers) now passes through our service worker. All of those requests can now be intercepted, analyzed, and even manipulated. ☺

```
self.addEventListener("fetch", function(event) {  
  if (event.request.url.includes("bootstrap.min.css")) {  
    event.respondWith(  
      new Response(  
        "div {background: green;}",  
        { headers: { "Content-Type": "text/css" } }  
      )  
    );  
  }  
});
```

# Service Worker

- Respond to requests with content from the web:

```
self.addEventListener("fetch", function(event) {  
  if (event.request.url.includes("/img/logo.png")) {  
    event.respondWith(  
      fetch("/img/logo-rotated.png")  
    );  
  }  
});
```

# Fetch API

- Documentation: [https://developer.mozilla.org/en-US/docs/Web/API/Fetch\\_API](https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API)
- Provides an interface for fetching resources (including across the network).
- Similar to **XMLHttpRequest**, but the new API provides a more powerful and flexible feature set.
- Moreover, **XMLHttpRequest** is **not** available in ServiceWorkers.

# Fetch API

```
// Fetch by URL  
fetch("/img/logo.png");  
// Fetch by the URL in the request object  
fetch(event.request.url);  
// Fetch by passing a request object.  
// In addition to the URL, the request object may contain additional  
headers,  
// form data, etc.  
fetch(event.request);
```

- The first argument in fetch is mandatory and can contain either a request object, or a string with a relative or absolute URL

# Fetch API

- The second argument is **optional** and can contain an object with options for the request.

```
return fetch(url, { // Default options are marked with *
  method: "POST", // *GET, POST, PUT, DELETE, etc.
  mode: "cors", // no-cors, cors, *same-origin
  cache: "no-cache", // *default, no-cache, reload, force-cache, only-if-cached
  credentials: "same-origin", // include, *same-origin, omit
  headers: {
    "Content-Type": "application/json; charset=utf-8",
    // "Content-Type": "application/x-www-form-urlencoded",
  },
  redirect: "follow", // manual, *follow, error
  referrer: "no-referrer", // no-referrer, *client
  body: JSON.stringify(data), // body data type must match "Content-Type" header
})
.then(response => response.json()); // parses response to JSON
```

# Service Worker - Capturing Offline Requests

- Handle the communication error using "catch"
- Simple "text" response:

```
self.addEventListener("fetch", function(event) {  
  event.respondWith(  
    fetch(event.request).catch(function() {  
      return new Response(  
        "There seems to be a problem with your connection.\n"  
      );  
    })  
  );  
});
```

# Service Worker - Capturing Offline Requests

```
var responseContent =  
"<html><body>" +  
"<style>body {text-align: center; background-color: #333; color: #eee; }" +  
"</style>" +  
"<h1>Progressive Web Apps</h1>" +  
"<p>There seems to be a problem with your connection.</p>" +  
"</body></html>;  
  
self.addEventListener("fetch", function(event) {  
  event.respondWith(  
    fetch(event.request).catch(function() {  
      return new Response(  
        responseContent,  
        {headers: {"Content-Type": "text/html"} }  
      );  
    })  
  );  
});
```

## Demo

- Check what happens if we remove the headers
  
- Note:
  - Most web servers are configured to automatically serve most common file types with the correct headers automatically.
  - When a server sends an HTML file, it constructs a response that contains both the HTML, as well as many headers, including a Content-Type header letting the browser know what to do with the response.
  - Because we are constructing a response from scratch, it is up to us to set the correct headers.



# Service Worker

- If we modify the Service Worker, the changes **don't immediately** take effect after refreshing the browser. This is because the **old service worker** is still the *active* one, while your new service worker remains in a *waiting* state until the old one is no longer controlling the page.
- To ease development, we can tell the browser to let new service workers take immediate control of the page. In Chrome, this can be done by opening the Application tab of the developer tools, and under the Service Workers section enabling “Update on reload”. This makes sure that each time we change the service worker and refresh the page, the new service worker will immediately take control of the page.

# Progressive Web Apps

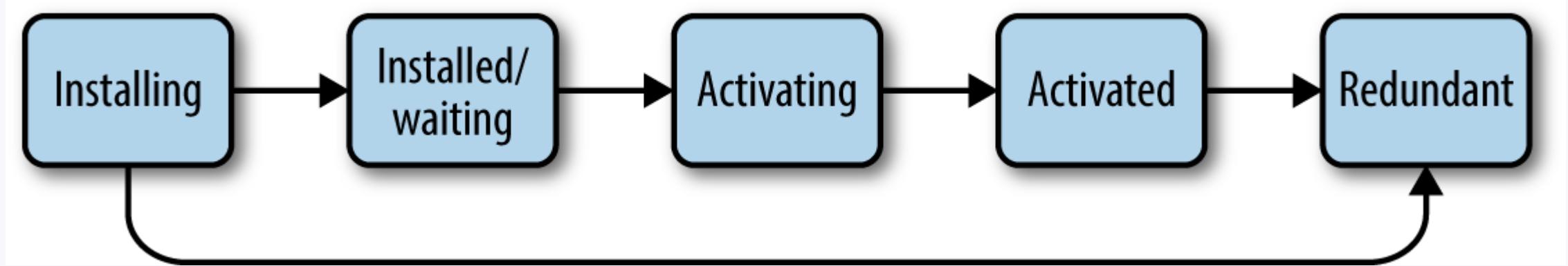
## Service Worker

The screenshot shows the Chrome DevTools interface with the Application tab selected. On the left, the sidebar lists 'Application' (Manifest, Service Workers, Clear storage), 'Storage' (Local Storage, Session Storage, IndexedDB, Web SQL, Cookies), and 'Cache' (Cache Storage, Application Cache). The main panel is titled 'Service Workers' and shows a list for '127.0.0.1'. It details a service worker registered from 'serviceworker.js' received on '12/4/2018, 9:49:11 AM'. The status is 'activated and is running'. A client is listed as 'http://127.0.0.1:5500/index.html'. There are buttons for 'Push' (with input field 'Test push message from DevTools.') and 'Sync' (with input field 'test-tag-from-devtools'). Top navigation tabs include Elements, Console, Sources, Network, Performance, Memory, Application (selected), Security, and Audits. Status indicators show 1 error and 1 warning.

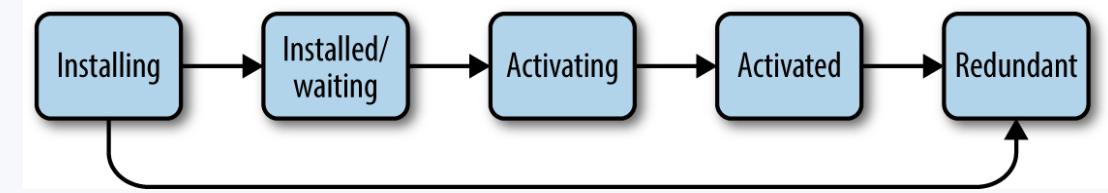
Turning on **Update on reload**

# Service Worker LifeCycle

- The lifecycle of a service worker after being downloaded:

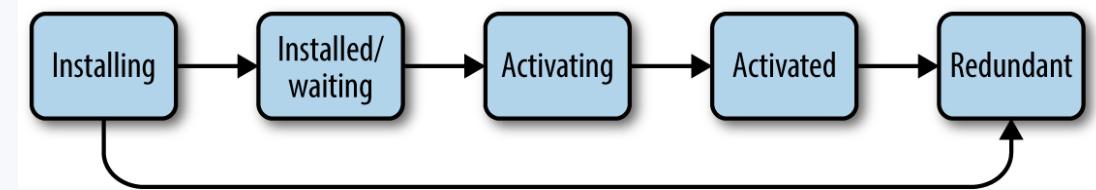


# Service Worker LifeCycle



- Download
  - the service worker is immediately downloaded when a user first accesses a service worker–controlled site/page.

# Service Worker LifeCycle

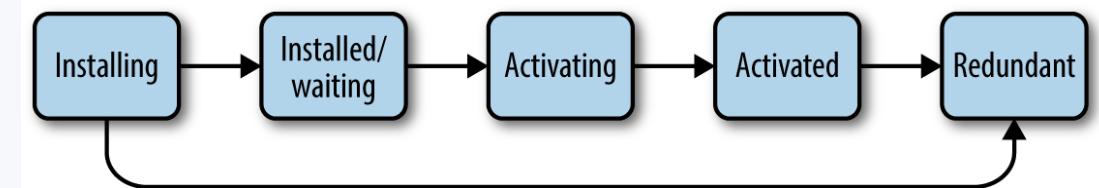


- **Installing state**

- When a new service worker is registered using `navigator.serviceWorker.register`, the JavaScript is downloaded, parsed, and enters the **installing** state.
- If the installation succeeds, the service worker will proceed to the **installed** state.
- The state can be extended by calling `event.waitUntil()` and passing it a promise.

```
self.addEventListener('install', function(e) {  
  e.waitUntil(  
    caches.open(cacheName).then(function(cache) {  
      return cache.addAll(filesToCache);  
    })  
  );  
});
```

# Service Worker LifeCycle



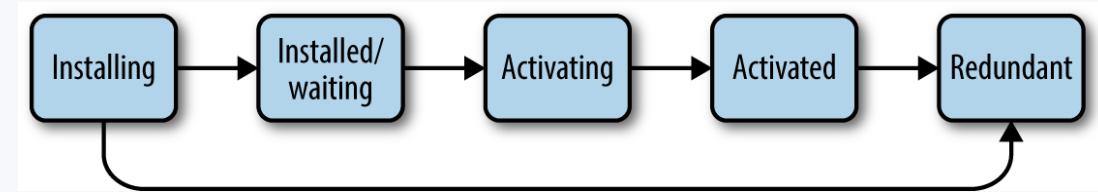
- If an error occurs during installation, the script will instead be moved to the redundant state.

```
self.addEventListener("install", function() {
  console.log("install");
  throw 'error';
});
```

A screenshot of the Chrome DevTools Application tab, specifically the Service Workers section. The tab bar at the top includes Sources, Network, Performance, Memory, Application (which is underlined), Security, and Audits. Below the tab bar, there are three checkboxes: Offline, Update on reload, and Bypass for network. The main area shows a single service worker entry for '127.0.0.1 - deleted'. The entry details are as follows:

- Source: [serviceworker.js](#) (with a red 'x' icon and the number 10)
- Received: 12/17/2018, 10:54:46 PM
- Status: ● #5507 is redundant
- Push: Test push message from DevTools. (with a Push button)
- Sync: test-tag-from-devtools (with a Sync button)

# Service Worker LifeCycle



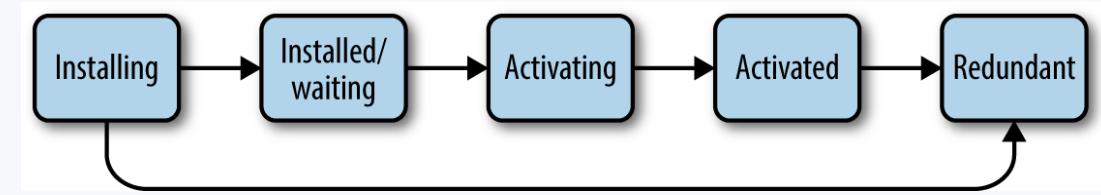
- Installed/waiting state

- Once a service worker has successfully installed, it enters the **installed state**.
- It will then immediately move on to the **activating state** unless another active service worker is currently controlling this app, in which case it will remain **waiting**.

The screenshot shows the Chrome DevTools Application tab for the URL 127.0.0.1. The left sidebar has sections for Application (Manifest, Service Workers, Clear storage), Storage (Local Storage, Session Storage, IndexedDB, Web SQL), and Network. The right pane shows the Service Workers panel with the following details:

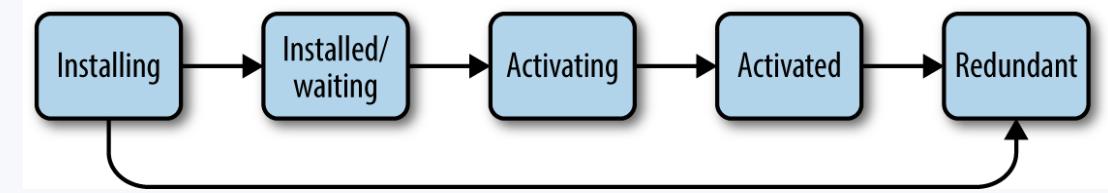
- Source: [serviceworker.js](#)
- Received: 12/17/2018, 11:07:59 PM
- Status:
  - #5509 activated and is running [stop](#)
  - #5511 waiting to activate [skipWaiting](#)
- Received: 12/17/2018, 11:08:15 PM
- Clients: <http://127.0.0.1:5500/index.html> [focus](#)

# Service Worker LifeCycle



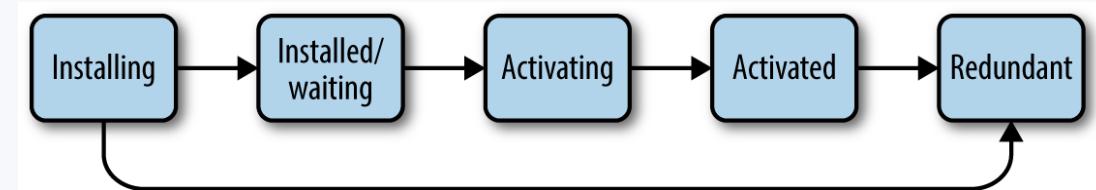
- It is only move to the **activating state** when there are no longer any pages loaded that are still using the old service worker.
- **Why?** Imagine a scenario in which you release a new version of a service worker, and this service worker's install event deletes userdata.json from the cache, adds users.json instead, and changes the fetch event to return the new file when user data is requested. If multiple service workers controlled different pages, the ones controlled by the old service worker might look for the old userdata.json file in the cache after it was removed, causing your app to break.
- Activation can happen sooner using `ServiceWorkerGlobalScope.skipWaiting()` and existing pages can be claimed by the active worker using `Clients.claim()`.

# Service Worker LifeCycle



- **Activating state**
  - Before a service worker becomes active and takes control of the app, the `activate` event is triggered.
  - The **activating state** can be extended by calling `event.waitUntil()` and passing it a promise.

# Service Worker LifeCycle

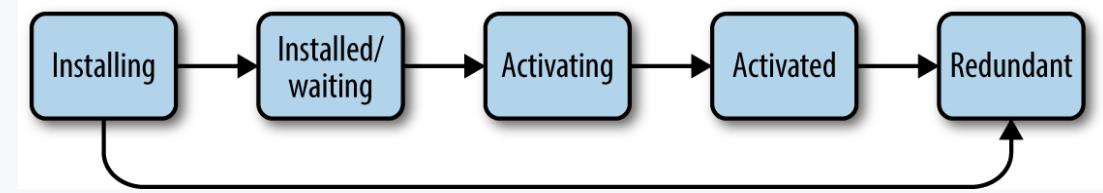


- Activating state

- Before a service worker becomes active and takes control of the app, the `activate` event is triggered.
- The **activating state** can be extended by calling `event.waitUntil()` and passing it a promise.

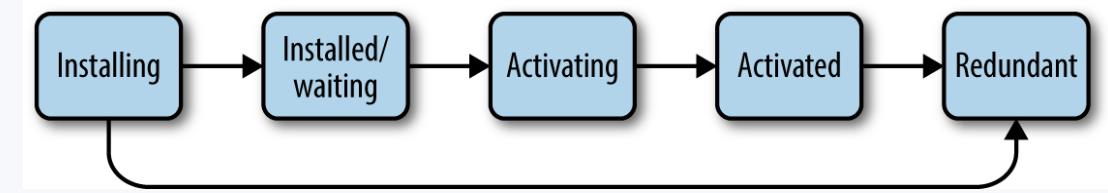
```
self.addEventListener('activate', function(e) {  
  e.waitUntil(  
    caches.keys().then(function(keyList) {  
      return Promise.all(keyList.map(function(key) {  
        if (key !== cacheName) {  
          return caches.delete(key);  
        }  
      }));  
    })  
  );  
  return self.clients.claim();  
});
```

# Service Worker LifeCycle



- Activated state
  - Once a service worker is activated, it is ready to take control of the page and listen to functional events (such as `fetch`).
  - Note: a service worker can only take control of pages before they start loading. This means that pages that began loading before the service worker became active cannot be controlled by it.

# Service Worker LifeCycle



- Redundant state
  - Service workers that failed during registration, or installation, or were replaced by newer versions, are placed in the **redundant state**. Service workers in this state no longer have any effect on your app.

# CacheStorage API

- Documentation:
  - <https://developers.google.com/web/fundamentals/instant-and-offline/web-storage/cache-api>
- The **Cache API** was created to enable Service Workers to cache network requests so that they can provide appropriate responses even while offline. However, the API can also be used as a general storage mechanism.

# CacheStorage API

- How to check whether the API is available?

```
const cacheAvailable = 'caches' in self;
```

- Creating and opening a cache

```
caches.open('my-cache').then((cache) => {
  // do something with cache...
});
```

# CacheStorage API

- Retrieving from a cache using `cache.match`

```
cache.match(request).then((response) => console.log(request, response));
```

- Retrieving all matching responses using `cache.matchAll`.

```
cache.matchAll(request).then((responses) => {
  console.log(`There are ${responses.length} matching responses.`);
});
```

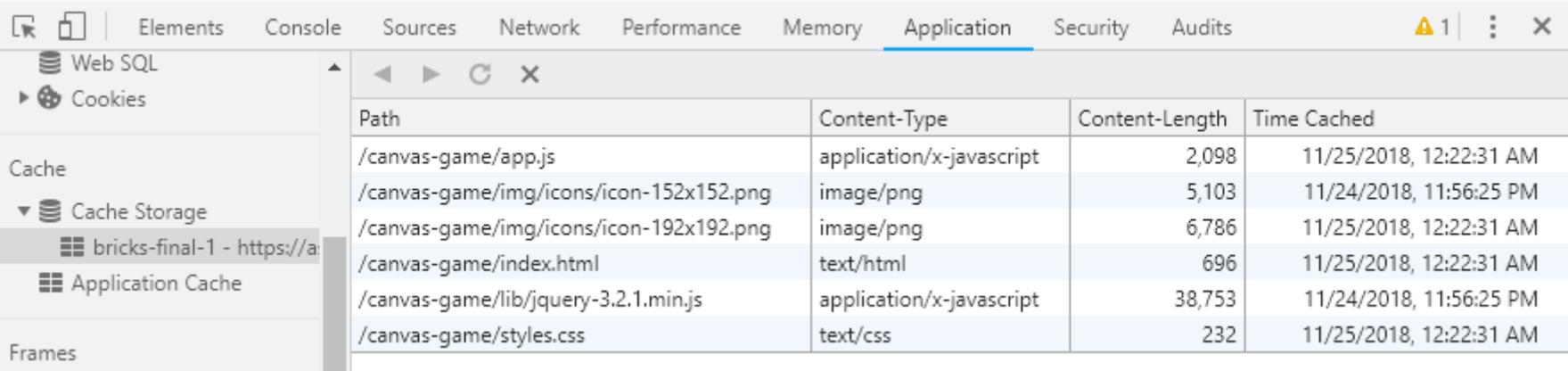
# CacheStorage API

- Adding to a cache
  - There are three ways to add an item to a cache - `put`, `add` and `addAll`. All three methods return a Promise.
  - `cache.put`

```
cache.put('/test.json', new Response('{"foo": "bar"}'));
```
  - `cache.add`
  - `cache.addAll`

# CacheStorage API

```
let cacheName = 'bricks-final-1';
let filesToCache = [
  'index.html',
  'app.js',
  'styles.css',
  'img/icons/icon-152x152.png',
  'img/icons/icon-192x192.png'
];
```



The screenshot shows the Google Chrome DevTools Application tab. On the left, there's a sidebar with sections like Web SQL, Cookies, Cache, Cache Storage (which is expanded to show 'bricks-final-1 - https://'), and Application Cache. The main area displays a table for 'bricks-final-1' with columns: Path, Content-Type, Content-Length, and Time Cached.

Path	Content-Type	Content-Length	Time Cached
/canvas-game/app.js	application/x-javascript	2,098	11/25/2018, 12:22:31 AM
/canvas-game/img/icons/icon-152x152.png	image/png	5,103	11/24/2018, 11:56:25 PM
/canvas-game/img/icons/icon-192x192.png	image/png	6,786	11/25/2018, 12:22:31 AM
/canvas-game/index.html	text/html	696	11/25/2018, 12:22:31 AM
/canvas-game/lib/jquery-3.2.1.min.js	application/x-javascript	38,753	11/24/2018, 11:56:25 PM
/canvas-game/styles.css	text/css	232	11/25/2018, 12:22:31 AM

Checking the **Cache Storage** in Google Chrome

# CacheStorage API

- Deleting an item

```
cache.delete(request);
```

- Where request can be a Request or a URL string.

# CacheStorage API

- Deleting a cache

```
caches.delete(key)
```

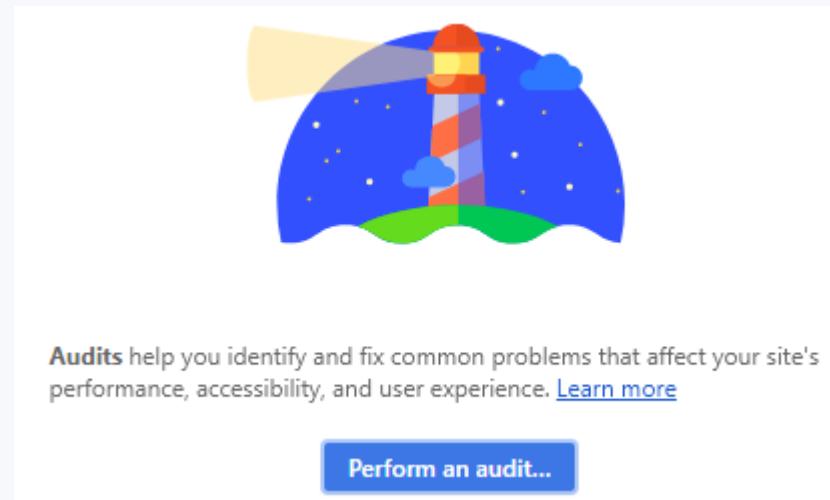
- The function returns a Promise that resolves to true if the cache existed and was deleted, or false otherwise.

# CacheStorage API

```
//2. Delete the old version of the cache
self.addEventListener('activate', function(e) {
  e.waitUntil(
    caches.keys().then(function(keyList) {
      return Promise.all(keyList.map(function(key) {
        if (key !== cacheName) {
          return caches.delete(key);
        }
      }));
    })
  );
  return self.clients.claim();
});
```

# Lighthouse

- an open-source, automated tool for improving the quality of Progressive Web Apps, that eliminates much of the manual testing that was previously required.
- integrated in Google Chrome.



# Is your App a Progressive Web App?

- Checklist: <https://developers.google.com/web/progressive-web-apps/checklist>
- Most of the checks are already included in Lighthouse ☺

## Examples



### Social Media - Twitter

- Windows Store Application
  - <https://www.microsoft.com/ro-ro/p/twitter/9wzdncrfj140>
- Twitter Lite for Android
  - <https://play.google.com/store/apps/details?id=com.twitter.android.lite&hl=en>

# JavaScript

# JavaScript

- Modern\_JavaScript.zip on <http://online.ase.ro>
- <http://jstherightway.org/>

# Web Storage API

- Documentation:
  - [https://developer.mozilla.org/en-US/docs/Web/API/Web\\_Storage\\_API](https://developer.mozilla.org/en-US/docs/Web/API/Web_Storage_API)